6 Searching for Structure in Point Data

"Through space the universe encompasses and swallows me up like an atom; through thought I comprehend the world." (Blaise Pascal)

We begin the third part of this book by addressing methods for exploring and quantifying structure in a multivariate distribution of points. One name for this kind of activity is *exploratory data analysis* (EDA). Given a sample of N points in D-dimensional space, there are three classes of problems that are frequently encountered in practice: density estimation, cluster finding, and statistical description of the observed structure. The space populated by points in the sample can be real physical space, or a space spanned by the measured quantities (attributes). For example, we can consider the distribution of sources in a multidimensional color space, or in a six-dimensional space spanned by three-dimensional positions and three-dimensional velocities.

To infer the pdf from a sample of data is known as density estimation. The same methodology is often called data smoothing. We have already encountered density estimation in the one-dimensional case when discussing histograms in §4.8 and §5.7.2, and in this chapter we extend it to multidimensional cases. Density estimation is one of the most critical components of extracting knowledge from data. For example, given a pdf estimated from point data, we can generate simulated distributions of data and compare them against observations. If we can identify regions of low probability within the pdf, we have a mechanism for the detection of unusual or anomalous sources. If our point data can be separated into subsamples using provided class labels, we can estimate the pdf for each subsample and use the resulting set of pdfs to classify new points: the probability that a new point belongs to each subsample/class is proportional to the pdf of each class evaluated at the position of the point (see §9.3.5). Density estimation relates directly to regression discussed in chapter 8 (where we simplify the problem to the prediction of a single variable from the pdf), and is at the heart of many of the classification procedures described in chapter 9. We discuss nonparametric and parametric methods for density estimation in §6.1–6.3.

Given a point data set, we can further ask whether it displays any structure (as opposed to a random distribution of points). Finding concentrations of multivariate points (or groups of sources) is known in astronomy as "clustering" (when a density estimate is available, clusters correspond to "overdensities"). Clusters can be defined

to be distinct objects (e.g., gravitationally bound clusters of galaxies), or loose groups of sources with common properties (e.g., the identification of quasars based on their color properties). Unsupervised clustering refers to cases where there is no prior information about the number and properties of clusters in data. Unsupervised classification, discussed in chapter 9, assigns to each cluster found by unsupervised clustering, a class based on additional information (e.g., clusters identified in color space might be assigned the labels "quasar" and "star" based on supplemental spectral data). Finding clusters in data is discussed in §6.4.

In some cases, such as when considering the distribution of sources in multi-dimensional color space, clusters can have specific physical meaning (e.g., hot stars, quasars, cold stars). On the other hand, in some applications, such as large-scale clustering of galaxies, clusters carry information only in a statistical sense. For example, we can test cosmological models of structure formation by comparing clustering statistics in observed and simulated data. Correlation functions are commonly used in astronomy for the statistical description of clustering, and are discussed in §6.5.

Data sets used in this chapter

In this chapter we use four data sets: a subset of the SDSS spectroscopic galaxy sample (§1.5.5), a set of SDSS stellar spectra with stellar parameter estimates (§1.5.7), SDSS single-epoch stellar photometry (§1.5.3), and the SDSS Standard Star Catalog from Stripe 82 (§1.5.8). The galaxy sample contains 8014 galaxies selected to be centered on the SDSS "Great Wall" (a filament of galaxies that is over 100 Mpc in extent; see [14]). These data comprise measures of the positions, luminosities and colors of the galaxies, and are used to illustrate density estimation and the spatial clustering of galaxies. The stellar spectra are used to derive measures of effective temperature, surface gravity, and two quantities that summarize chemical composition: metallicity (parametrized as [Fe/H]) and α-element abundance (parametrized as [α/Fe]). These measurements are used to illustrate clustering in multidimensional parameter space. The precise multiepoch averaged photometry from the Standard Star Catalog is used to demonstrate the performance of algorithms that account for measurement errors when estimating the underlying density (pdf) from the less precise single-epoch photometric data.

6.1. Nonparametric Density Estimation

In some sense chapters 3, 4, and 5 were about estimating the underlying density of the data, using parametric models. Chapter 3 discussed parametric models of probability density functions and chapters 4 and 5 discussed estimation of their parameters from frequentist and Bayesian perspectives. We now look at how to estimate a density *nonparametrically*, that is, without specifying a specific functional model. Real data rarely follow simple distributions—nonparametric methods are meant to capture every aspect of the density's shape. What we lose by taking this route is the convenience, relative computational simplicity (usually), and easy interpretability of parametric models.

The go-to method for nonparametric density estimation, that is, modeling of the underlying distribution, is the method of *kernel density estimation* (KDE).

While a very simple method in principle, it also comes with impressive theoretical properties.

6.1.1. Kernel Density Estimation

As a motivation for kernel density estimation, let us first reconsider the one-dimensional histograms introduced in §4.8. One problem with a standard histogram is the fact that the exact locations of the bins can make a difference, and yet it is not clear how to choose in advance where the bins should be placed (see §4.8.1). We illustrate this problem in the two top panels in figure 6.1. They show histograms constructed with an identical data set, and with identical bin widths, but with bins offset in x by 0.25. This offset leads to very different histograms and possible interpretations of the data: the difference between seeing it as a bimodal distribution vs. an extended flat distribution.

How can we improve on a basic histogram and avoid this problem? Each point within a histogram contributes one unit to the height of the histogram at the position of its bin. One possibility is to allow each point to have its own bin, rather than arranging the bins in a regular grid, and furthermore allow the bins to overlap. In essence, each point is replaced by a box of unit height and some predefined width. The result is the distribution shown in the middle-left panel of figure 6.1. This distribution does not require a specific choice of bin boundaries—the data drive the bin positioning—and does a much better job of showing the bimodal character of the underlying distribution than the histogram shown in the top-right panel.

The above simple recipe for producing a histogram is an example of kernel density estimation. Here the kernel is a top-hat distribution centered on each individual point. It can be shown theoretically that this kernel density estimator (KDE) is a better estimator of the density than the ordinary histogram (see Wass10). However, as is discernible from the middle-left panel of figure 6.1, the rectangular kernel does not lead to a very smooth distribution and can even display suspicious spikes. For this reason, other kernels, for example Gaussians, are often used. The remaining panels of figure 6.1 show the kernel density estimate, described below, of the same data but now using Gaussian kernels of different widths. Using too narrow a kernel (middle-right panel) leads to a noisy distribution, while using too wide a kernel (bottom-left panel) leads to excessive smoothing and washing out of information. A well-tuned kernel (bottom-right panel) can lead to accurate estimation of the underlying distribution; the choice of kernel width is discussed below.

Given a set of measurements $\{x_i\}$, the kernel density estimator (i.e., an estimator of the underlying pdf) at an arbitrary position x is defined as

$$\widehat{f}_N(x) = \frac{1}{Nh^D} \sum_{i=1}^{N} K\left(\frac{d(x, x_i)}{h}\right), \tag{6.1}$$

where $K(u)$ is the *kernel function* and h is known as the bandwidth (which defines the size of the kernel). The local density is estimated as a weighted mean of all points, where the weights are specified via $K(u)$ and typically decrease with distance $d(x, x_i)$. Alternatively, KDE can be viewed as replacing each point with a "cloud" described by $K(u)$. The kernel function $K(u)$ can be any smooth function that is positive at all points ($K(u) \geq 0$), normalizes to unity ($\int K(u)\, du = 1$), has a mean of zero

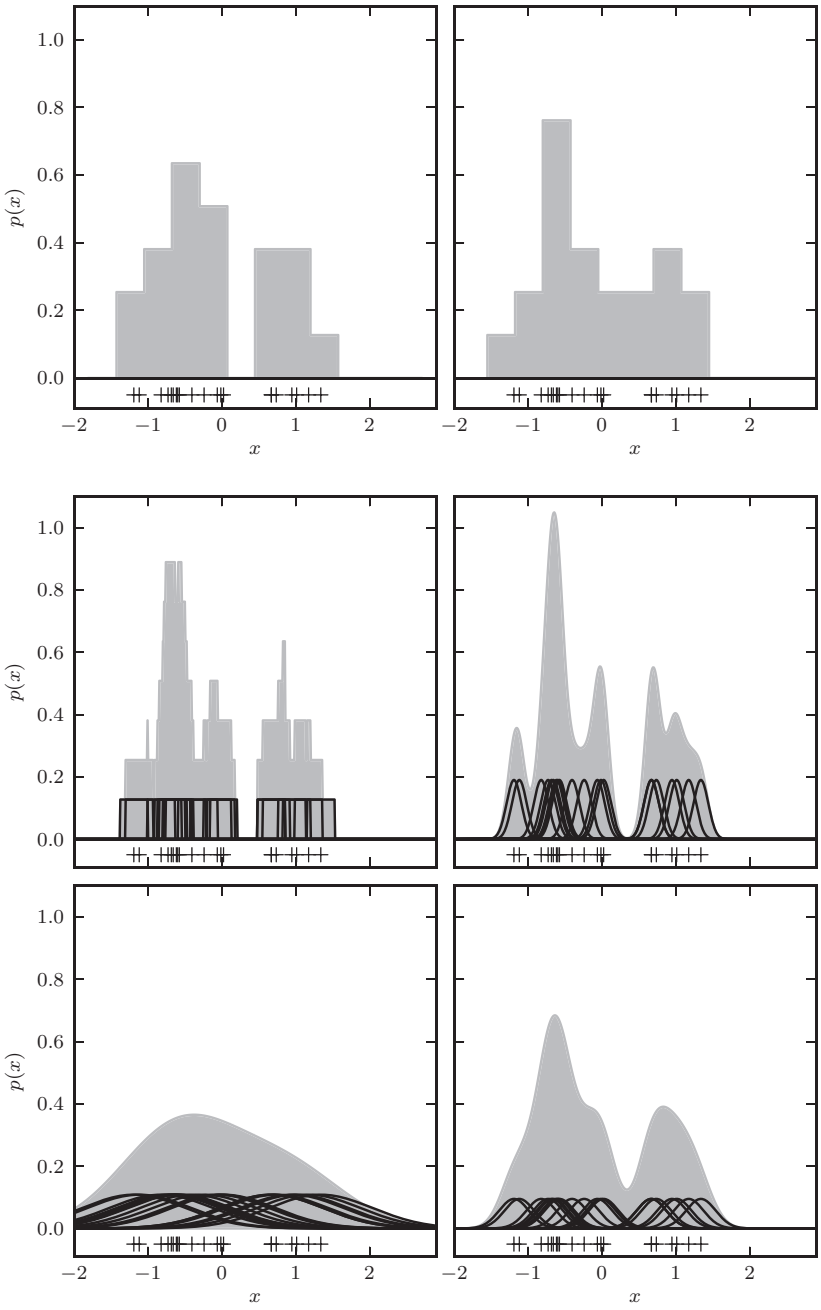


Figure 6.1. Density estimation using histograms and kernels. The top panels show two histogram representations of the same data (shown by plus signs in the bottom of each panel) using the same bin width, but with the bin centers of the histograms offset by 0.25. The middle-left panel shows an adaptive histogram where each bin is centered on an individual point and these bins can overlap. This adaptive representation preserves the bimodality of the data. The remaining panels show kernel density estimation using Gaussian kernels with different bandwidths, increasing from the middle-right panel to the bottom-right, and with the largest bandwidth in the bottom-left panel. The trade-off of variance for bias becomes apparent as the bandwidth of the kernels increases.

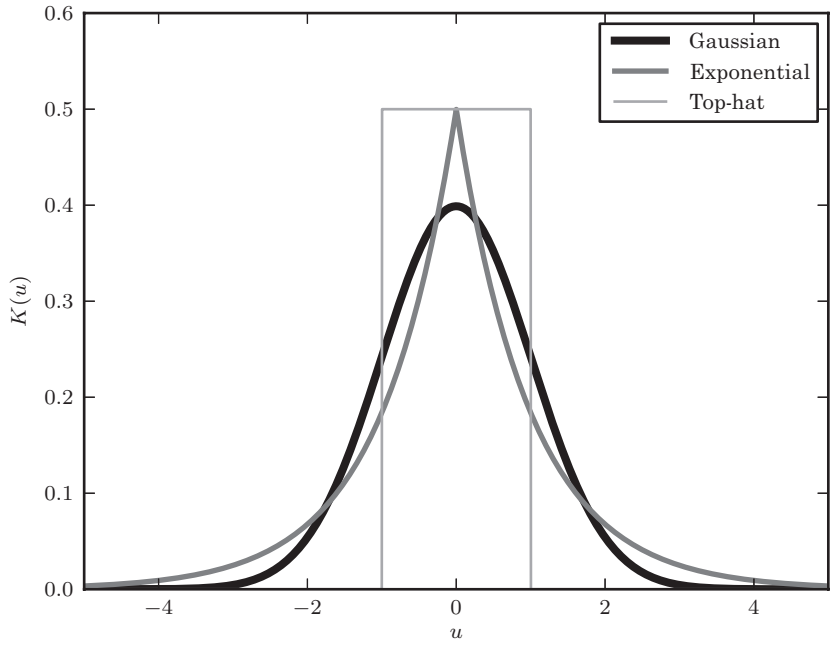


Figure 6.2. A comparison of the three kernels used for density estimation in figure 6.3: the Gaussian kernel (eq. 6.2), the top-hat kernel (eq. 6.3), and the exponential kernel (eq. 6.4).

($\int uK(u)\,du = 0$), and has a variance ($\sigma_K^2 = \int u^2 K(u)\,du$) greater than zero. An often-used kernel is the Gaussian kernel,

$$K(u) = \frac{1}{(2\pi)^{D/2}} e^{-u^2/2}, \tag{6.2}$$

where D is the number of dimensions of the parameter space and $u = d(x, x_i)/h$. Other kernels that can be useful are the top-hat (box) kernel,

$$K(u) = \begin{cases} \frac{1}{V_D(1)} & \text{if } u \leq 1, \\ 0 & \text{if } u > 1, \end{cases} \tag{6.3}$$

and the exponential kernel,

$$K(u) = \frac{1}{D!\, V_D(1)} e^{-|u|}, \tag{6.4}$$

where $V_D(r)$ is the volume of a D-dimensional hypersphere of radius r (see eq. 7.3). A comparison of the Gaussian, exponential, and top-hat kernels is shown in figure 6.2.

Selecting the KDE bandwidth using cross-validation

Both histograms and KDE do, in fact, have a parameter: the kernel or bin width. The proper choice of this parameter is critical, much more so than the choice of a specific kernel, particularly when the data set is large; see [41]. We will now show a rigorous

procedure for choosing the optimal kernel width in KDE (which can also be applied to finding the optimal bin width for a histogram).

Cross-validation can be used for any cost function (see §8.11); we just have to be able to evaluate the cost on *out-of-sample* data (i.e., points not in the training set). If we consider the likelihood cost for KDE, for which we have the leave-one-out *likelihood cross-validation*, then the cost is simply the sum over all points in the data set (i.e., $i = 1, \ldots, N$) of the log of the likelihood of the density, where the density, $\widehat{f}_{h,-i}(x_i)$, is estimated leaving out the ith data point. This can be written as

$$\mathrm{CV}_l(h) = \frac{1}{N}\sum_{i=1}^{N} \log \widehat{f}_{h,-i}(x_i), \tag{6.5}$$

and, by minimizing $C\,V_l(h)$ as a function of bandwidth, we can optimize for the width of the kernel h.

An alternative to likelihood cross-validation is to use the mean integrated square error (MISE), introduced in eq. 4.14, as the cost function. To determine the value of h that minimizes the MISE we can write

$$\int (\widehat{f}_h - f)^2 = \int \widehat{f}_h^2 - 2\int \widehat{f}_h f + \int f^2. \tag{6.6}$$

As before, the first term can be obtained analytically, and the last term does not depend on h. For the second term we have expectation value

$$\mathbb{E}\left[\int \widehat{f}_h(x) f(x)\, dx\right] = \mathbb{E}\left[\frac{1}{N}\sum_{i=1}^{N} \widehat{f}_{h,-1}(x)\right]. \tag{6.7}$$

This motivates the L_2 *cross-validation* score:

$$\mathrm{CV}_{L_2}(h) = \int \widehat{f}_h^2 - 2\frac{1}{N}\sum_{i=1}^{N} \widehat{f}_{h,-i}(x_i) \tag{6.8}$$

since $\mathbb{E}[\mathrm{CV}_{L_2}(h) + \int f^2] = \mathbb{E}[\mathrm{MISE}(\widehat{f}_h)]$.

The optimal KDE bandwidth decreases at the rate $\mathcal{O}(N^{-1/5})$ (in a one-dimensional problem), and the error of the KDE using the optimal bandwidth converges at the rate $\mathcal{O}(N^{-4/5})$; it can be shown that histograms converge at a rate $\mathcal{O}(N^{-2/3})$; see [35]. KDE is, therefore, theoretically superior to the histogram as an estimator of the density. It can also be shown that there does not exist a density estimator that converges faster than $\mathcal{O}(N^{-4/5})$ (see Wass10).

Ideally we would select a kernel that has h as small as possible. If h becomes too small we increase the variance of the density estimation. If h is too large then the variance decreases but at the expense of the bias in the derived density. The optimal kernel function, in terms of minimum variance, turns out to be

$$K(x) = \frac{3}{4}\left(1 - x^2\right) \tag{6.9}$$

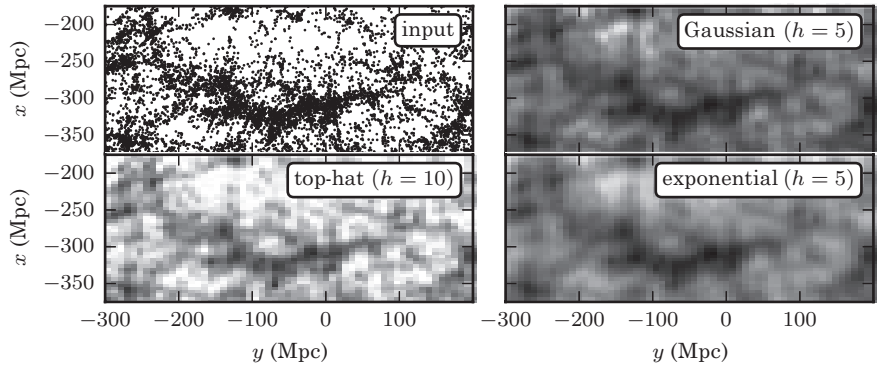


Figure 6.3. Kernel density estimation for galaxies within the SDSS "Great Wall." The top-left panel shows points that are galaxies, projected by their spatial locations (right ascension and distance determined from redshift measurement) onto the equatorial plane (declination $\sim 0^\circ$). The remaining panels show estimates of the density of these points using kernel density estimation with a Gaussian kernel (upper right), a top-hat kernel (lower left), and an exponential kernel (lower right). Compare also to figure 6.4.

for $|x| \leq 1$ and 0 otherwise; see [37]. This function is called the *Epanechnikov kernel*.

AstroML contains an implementation of kernel density estimation in D dimensions using the above kernels:

```
import numpy as np
from astroML.density_estimation import KDE

X = np.random.normal(size=(1000, 2)) # 1000 points
    # in 2 dims
kde = KDE('gaussian', h=0.1) # select the gaussian
      # kernel
kde.fit(X) # fit the model to the data
dens = kde.eval(X) # evaluate the model at the data
```

There are several choices for the kernel. For more information and examples, see the AstroML documentation or the code associated with the figures in this chapter.

Figure 6.3 shows an example of KDE applied to a two-dimensional data set: a sample of galaxies centered on the SDSS "Great Wall." The distribution of points (galaxies) shown in the top-left panel is used to estimate the "smooth" underlying distribution using three types of kernels. The top-hat kernel (bottom-left panel) is the most "spread out" of the kernels, and its imprint on the resulting distribution is apparent, especially in underdense regions. Between the Gaussian and exponential kernels, the exponential is more sharply peaked and has wider tails, but both recover similar features in the distribution. For a comparison of other density estimation methods for essentially the same data set, see [12].

Computation of kernel density estimates To obtain the height of the density estimate at a single query point (position) x, we must sum over N kernel functions. For many (say, $\mathcal{O}(N)$) queries, when N grows very large this brute-force approach can lead to very long ($\mathcal{O}(N^2)$) computation time. Because the kernels usually have a limited bandwidth h, points $\mathbf{x}_i$ with $|\mathbf{x_i} - \mathbf{x}| \gg h$ contribute a negligible amount to the density at a point $\mathbf{x}$, and the bulk of the contribution comes from neighboring points. However, a simplistic cutoff approach in an attempt to directly copy the nearest-neighbor algorithm we discussed in §2.5.2 leads to potentially large, unquantified errors in the estimate, defeating the purpose of an accurate nonparametric density estimator. More principled approaches were introduced in [15] but improved upon in subsequent research: for the highest accuracy and speed in low to moderate dimensionalities see the dual-tree fast Gauss transforms in [22, 24], and in higher dimensionalities see [23]. Ram et al. [33] showed rigorously that such algorithms reduce the runtime of a single query of KDE from the naive $\mathcal{O}(N)$ to $\mathcal{O}(\log N)$, and for $\mathcal{O}(N)$ queries from the naive $\mathcal{O}(N^2)$ to $\mathcal{O}(N)$. An example of the application of such algorithms in astronomy was shown in [3]. A parallelization of such algorithms is shown in [25]. A overview of tree algorithms for KDE and other problems can be found in chapter 21 of WSAS.

6.1.2. KDE with Measurement Errors

Suppose now that the points (i.e., their coordinates) are measured with some error σ. We begin with the simple one-dimensional case with homoscedastic errors. Assume that the data is drawn from the true pdf $h(x)$, and the error is described by the distribution $g(x|\sigma)$. Then the observed distribution $f(x)$ is given by the convolution (see §3.44)

$$f(x) = (h \star g)(x) = \int_{-\infty}^{\infty} h(x')g(x - x')\,dx'. \tag{6.10}$$

This suggests that in order to obtain the underlying noise-free density $h(x)$, we can obtain an estimate $f(x)$ from the noisy data first, and then "deconvolve" the noise pdf. The nonparametric method of *deconvolution KDE* does precisely this; see [10, 38]. According to the convolution theorem, a convolution in real space corresponds to a product in Fourier space (see §10.2.2 for details). Because of this, deconvolution KDE can be computed using the following steps:

1. Find the kernel density estimate of the observed data, $f(x)$, and compute the Fourier transform $F(k)$.
2. Compute the Fourier transform $G(k)$ of the noise distribution $g(x)$.
3. From eq. 6.10 and the convolution theorem, the Fourier transform of the true distribution $h(x)$ is given by $H(k) = F(k)/G(k)$. The underlying noise-free pdf $h(x)$ can be computed via the inverse Fourier transform of $H(k)$.

For certain kernels $K(x)$ and certain noise distributions $g(x)$, this deconvolution can be performed analytically and the result becomes another modified kernel, called the *deconvolved kernel*. Examples of kernel and noise forms which have these properties can be found in [10, 38]. Here we will describe one example of a D-dimensional version of this method, where the noise scale is assumed to be heteroscedastic and

dependent on the dimension. The noise model leads to an analytic treatment of the deconvolved kernel. We will assume that the noise is distributed according to the multivariate exponential

$$g(\mathbf{x}) = \frac{1}{\sqrt{2^D}\sigma_1\sigma_2\ldots\sigma_D} \exp\left[-\sqrt{2}\left(\frac{|x_1|}{\sigma_1} + \frac{|x_2|}{\sigma_2} + \cdots + \frac{|x_D|}{\sigma_D}\right)\right], \tag{6.11}$$

where the σ_i represent the standard deviations in each dimension. We will assume that $(\sigma_1, \ldots, \sigma_D)$ are known for each data point. For the case of a Gaussian kernel function, the deconvolution kernel is then

$$K_{h,\sigma}(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^D}} \exp(-|\mathbf{x}|^2/2) \prod_i \left(1 - \frac{\sigma_i^2}{2h^2}(x_i^2 - 1)\right). \tag{6.12}$$

This deconvolution kernel can then be used in place of the kernels discussed in §6.1.1 above, noting the additional dependence on the error σ of each point.

6.1.3. Extensions and Related Methods

The idea of kernel density estimation can be extended to other tasks, including classification (kernel discriminant analysis, §9.3.5), regression (kernel regression, §8.5), and conditional density estimation (kernel conditional density estimation, §3.1.3). Some of the ideas that have been developed to make kernel regression highly accurate, discussed in 8.5, can be brought back to density estimation, including the idea of using variable bandwidths, in which each data point can have its own kernel width.

6.2. Nearest-Neighbor Density Estimation

Another often used and simple density estimation technique is based on the distribution of nearest neighbors. For each point (e.g., a pixel location on the two-dimensional grid) we can find the distance to the Kth-nearest neighbor, d_K. In this method, originally proposed in an astronomical context by Dressler et al. [11], the implied point density at an arbitrary position x is estimated as

$$\widehat{f}_K(x) = \frac{K}{V_D(d_K)}, \tag{6.13}$$

where the volume V_D is evaluated according to the problem dimensionality, D (e.g., for $D = 2$, $V_2 = \pi d^2$; for $D = 3$, $V_3 = 4\pi d^3/3$; for higher dimensions, see eq. 7.3). The simplicity of this estimator is a consequence of the assumption that the underlying density field is locally constant. In practice, the method is even simpler because one can compute

$$\widehat{f}_K(x) = \frac{C}{d_K^D}, \tag{6.14}$$

and evaluate the scaling factor C at the end by requiring that the sum of the product of $\widehat{f}_K(x)$ and pixel volume is equal to the total number of data points. The error in $\widehat{f}_K(x)$ is $\sigma_f = K^{1/2}/V_D(d_K)$, and the fractional (relative) error is $\sigma_f/\widehat{f} = 1/K^{1/2}$. Therefore, the fractional accuracy increases with K at the expense of the spatial resolution (the effective resolution scales with $K^{1/D}$). In practice, K should be at least 5 because the estimator is biased and has a large variance for smaller K; see [7].

This general method can be improved (the error in $\widehat{f}$ can be decreased without a degradation in the spatial resolution, or alternatively the resolution can be increased without increasing the error in $\widehat{f}$) by considering distances to *all* K nearest neighbors instead of only the distance to the Kth-nearest neighbor; see [18]. Given distances to all K neighbors, $d_i, i = 1, \ldots, K$,

$$\widehat{f}_K(x) = \frac{C}{\sum_{i=1}^{K} d_i^D}. \tag{6.15}$$

Derivation of eq. 6.15 is based on Bayesian analysis, as described in [18]. The proper normalization when computing local density without regard to overall mean density is

$$C = \frac{K(K+1)}{2V_D(1)}. \tag{6.16}$$

When searching for local overdensities in the case of sparse data, eq. 6.15 is superior to eq. 6.14; in the constant density case, both methods have similar statistical power. For an application in an astronomical setting, see [36].

AstroML implements nearest-neighbor density estimation using a fast ball-tree algorithm. This can be accomplished as follows:

```
import numpy as np
from astroML.density_estimation import
   KNeighborsDensity

X = np.random.normal(size=(1000, 2)) # 1000 points
    # in 2 dims
knd = KNeighborsDensity("bayesian", 10) # bayesian
      # method, 10 nbrs
knd.fit(X) # fit the model to the data
dens = knd.eval(X) # evaluate the model at the data
```

The method can be either `"simple"` to use eq. 6.14, or `"bayesian"` to use eq. 6.15. See the AstroML documentation or the code associated with figure 6.4 for further examples.

Figure 6.4 compares density estimation using the Gaussian kernel with a bandwidth of 5 Mpc and using the nearest-neighbor method (eq. 6.15) with $K = 5$ and $K = 40$ for the same sample of galaxies as shown in figure 6.3. For small K the

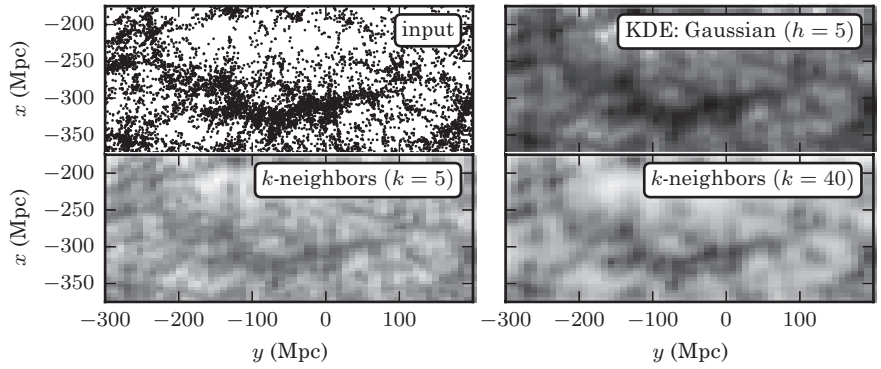


Figure 6.4. Density estimation for galaxies within the SDSS "Great Wall." The upper-left panel shows points that are galaxies, projected by their spatial locations onto the equatorial plane (declination $\sim 0^\circ$). The remaining panels show estimates of the density of these points using kernel density estimation (with a Gaussian kernel with width 5 Mpc), a K-nearest-neighbor estimator (eq. 6.15) optimized for a small-scale structure (with $K = 5$), and a K-nearest-neighbor estimator optimized for a large-scale structure (with $K = 40$).

fine structure in the galaxy distribution is preserved but at the cost of a larger variance in the density estimation. As K increases the density distribution becomes smoother, at the cost of additional bias in the other estimates.

Figure 6.5 compares Bayesian blocks, KDE, and nearest-neighbor density estimation for two one-dimensional data sets drawn from the same (relatively complicated) generating distribution (this is the same generated data set used previously in figure 5.21). The generating distribution includes several "peaks" that are described by the Cauchy distribution (§3.3.5). KDE and nearest-neighbor methods are much noisier than the Bayesian blocks method in the case of the smaller sample; for the larger sample all three methods produce similar results.

6.3. Parametric Density Estimation

KDE estimates the density of a set of points by affixing a kernel to each point in the data set. An alternative is to use fewer kernels, and fit for the kernel locations as well as the widths. This is known as a *mixture model*, and can be viewed in two ways: at one extreme, it is a density estimation model similar to KDE. In this case one is not concerned with the locations of individual clusters, but the contribution of the full set of clusters at any given point. At the other extreme, it is a clustering algorithm, where the location and size of each component is assumed to reflect some underlying property of the data.

6.3.1. Gaussian Mixture Model

The most common mixture model uses Gaussian components, and is called a *Gaussian mixture model* (GMM). A GMM models the underlying density (pdf) of points as a sum of Gaussians. We have already encountered one-dimensional mixtures of Gaussians in §4.4; in this section we extend those results to multiple

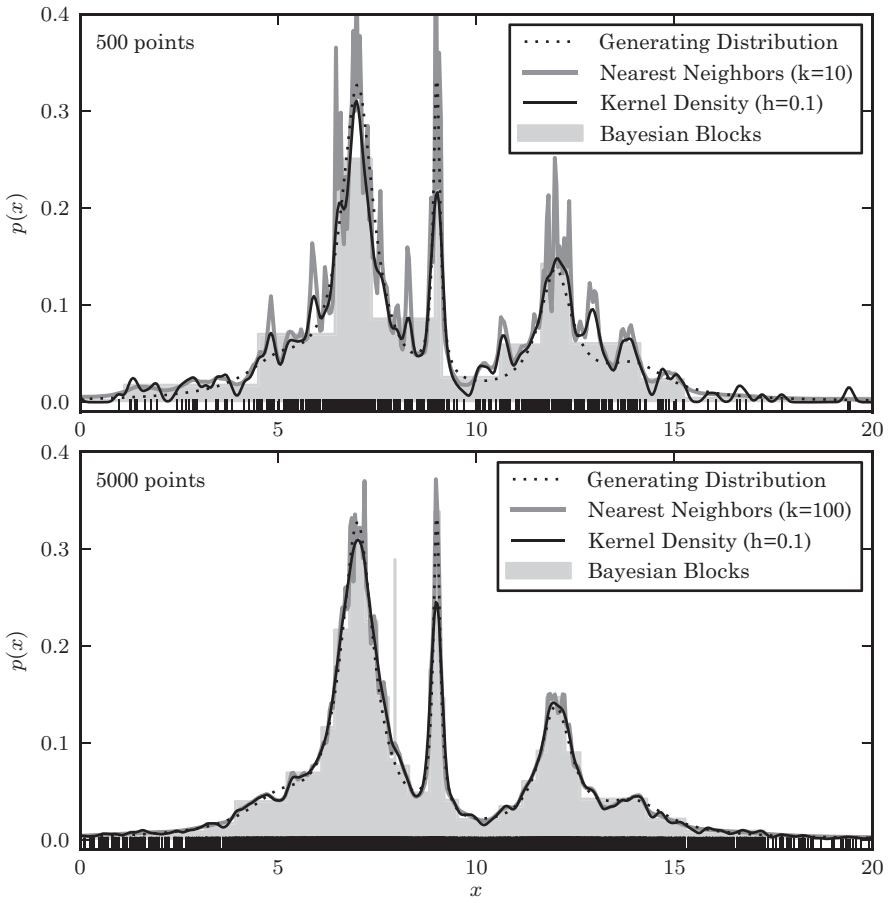


Figure 6.5. A comparison of different density estimation methods for two simulated one-dimensional data sets (cf. figure 5.21). The generating distribution is same in both cases and shown as the dotted line; the samples include 500 (top panel) and 5000 (bottom panel) data points (illustrated by vertical bars at the bottom of each panel). Density estimators are Bayesian blocks (§5.7.2), KDE (§6.1.1) and the nearest-neighbor method (eq. 6.15).

dimensions. Here the density of the points is given by (cf. eq. 4.18)

$$\rho(\mathbf{x}) = Np(\mathbf{x}) = N \sum_{j=1}^{M} \alpha_j \mathcal{N}(\mu_j, \Sigma_j), \tag{6.17}$$

where the model consists of M Gaussians with locations μ_j and covariances Σ_j. The likelihood of the data can be evaluated analogously to eq. 4.20. Thus there is not only a clear score that is being optimized, the log-likelihood, but this is a special case where that function is a generative model, that is, it is a full description of the data.

The optimization of this likelihood is more complicated in multiple dimensions than in one dimension, but the expectation maximization methods discussed in §4.4.3 can be readily applied in this situation; see [34]. We have already shown a simple example in one dimension for a toy data set (see figure 4.2). Here we will show

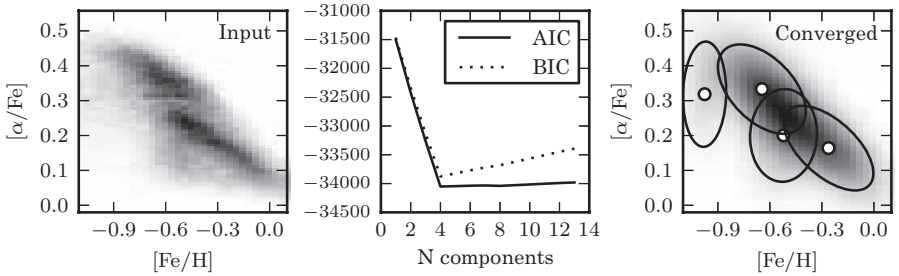


Figure 6.6. A two-dimensional mixture of Gaussians for the stellar metallicity data. The left panel shows the number density of stars as a function of two measures of their chemical composition: metallicity ([Fe/H]) and α-element abundance ([α/Fe]). The right panel shows the density estimated using mixtures of Gaussians together with the positions and covariances (2σ levels) of those Gaussians. The center panel compares the information criteria AIC and BIC (see §4.3.2 and §5.4.3).

an implementation of Gaussian mixture models for data sets in two dimensions, taken from real observations. In a later chapter, we will also apply this method to data in up to seven dimensions (see §10.3.4).

Scikit-learn includes an implementation of Gaussian mixture models in D dimensions:

```
import numpy as np
from sklearn.mixture import GMM

X = np.random.normal(size=(1000, 2)) # 1000 points
    # in 2 dims
gmm = GMM(3) # three component mixture
gmm.fit(X) # fit the model to the data
log_dens = gmm.score(X) # evaluate the log density
BIC = gmm.bic(X) # evaluate the BIC
```

For more involved examples, see the Scikit-learn documentation or the source code for figures in this chapter.

The left panel of figure 6.6 shows a Hess diagram (essentially a two-dimensional histogram) of the [Fe/H] vs. [α/Fe] metallicity for a subset of the SEGUE Stellar Parameters data (see §1.5.7). This diagram shows two distinct clusters in metallicity. For this reason, one may expect (or hope!) that the best-fit mixture model would contain two Gaussians, each containing one of those peaks. As the middle panel shows, this is not the case: the AIC and BIC (see §4.3.2) both favor models with four or more components. This is due to the fact that the components exist within a background, and the background level is such that a two-component model is insufficient to fully describe the data.

Following the BIC, we select $N = 4$ components, and plot the result in the rightmost panel. The reconstructed density is shown in grayscale and the positions of

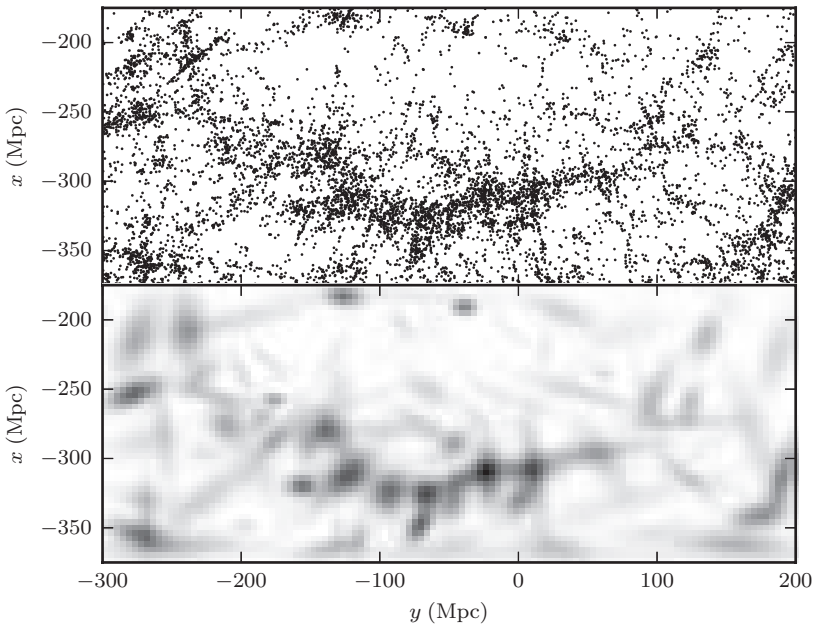


Figure 6.7. A two-dimensional mixture of 100 Gaussians (bottom) used to estimate the number density distribution of galaxies within the SDSS Great Wall (top). Compare to figures 6.3 and 6.4, where the density for the same distribution is computed using both kernel density and nearest-neighbor-based estimates.

the Gaussians in the model as solid ellipses. The two strongest components do indeed fall on the two peaks, where we expected them to lie. Even so, these two Gaussians do not completely separate the two clusters.

This is one of the common misunderstandings of Gaussian mixture models: the fact that the information criteria, such as BIC/AIC, prefer an N-component peak does not necessarily mean that there are N components. If the clusters in the input data are not near Gaussian, or if there is a strong background, the number of Gaussian components in the mixture will not generally correspond to the number of clusters in the data. On the other hand, if the goal is to simply describe the underlying pdf, many more components than suggested by BIC can be (and should be) used.

Figure 6.7 illustrates this point with the SDSS "Great Wall" data where we fit 100 Gaussians to the point distribution. While the underlying density representation is consistent with the distribution of galaxies and the positions of the Gaussians themselves correlate with the structure, there is not a one-to-one mapping between the Gaussians and the positions of clusters within the data. For these reasons, mixture models are often more appropriate when used as a density estimator as opposed to cluster identification (see, however, §10.3.4 for a higher-dimensional example of using GMM for clustering).

Figure 6.8 compares one-dimensional density estimation using Bayesian blocks, KDE, and a Gaussian mixture model using the same data sets as in figure 6.5. When the sample is small, the GMM solution with three components is favored by the BIC criterion. However, one of the components has a very large width ($\mu = 8, \sigma = 26$) and effectively acts as a nearly flat background. The reason for such a bad GMM

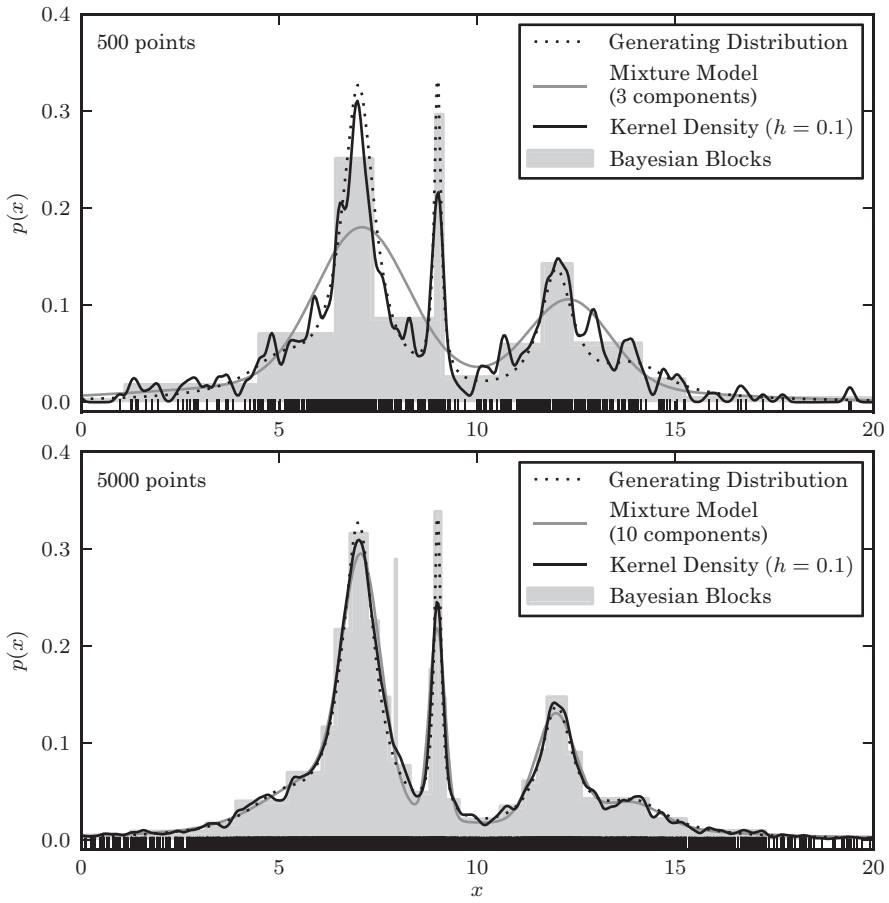


Figure 6.8. A comparison of different density estimation methods for two simulated one-dimensional data sets (same as in figure 6.5). Density estimators are Bayesian blocks (§5.7.2), KDE (§6.1.1), and a Gaussian mixture model. In the latter, the optimal number of Gaussian components is chosen using the BIC (eq. 5.35). In the top panel, GMM solution has three components but one of the components has a very large width and effectively acts as a nearly flat background.

performance (compared to Bayesian blocks and KDE which correctly identify the peak at $x \sim 9$) is the fact that individual "peaks" are generated using the Cauchy distribution: the wide third component is trying (hard!) to explain the wide tails. In the case of the larger sample, the BIC favors ten components and they obtain a similar level of performance to the other two methods.

BIC is a good tool to find how many statistically significant clusters are supported by the data. However, when density estimation is the only goal of the analysis (i.e., when individual components or clusters are not assigned any specific meaning) we can use any number of mixture components (e.g., when underlying density is very complex and hard to describe using a small number of Gaussian components). With a sufficiently large number of components, mixture models approach the flexibility of nonparametric density estimation methods.

Determining the number of components

Most mixture methods require that we specify the number of components as an input to the method. For those methods which are based on a score or error, determination of the number of components can be treated as a model selection problem like any other (see chapter 5), and thus be performed via cross-validation (as we did when finding optimal kernel bandwidth; see also §8.11), or using BIC/AIC criteria (§5.4.3). The hierarchical clustering method (§6.4.5) addresses this problem by finding clusterings at all possible scales.

It should be noted, however, that specifying the number of components (or clusters) is a relatively poorly posed question in astronomy. It is rare, despite the examples given in many machine learning texts, to find distinct, isolated and Gaussian clusters of data in an astronomical distribution. Almost all distributions are continuous. The number of clusters (and their positions) relates more to how well we can characterize the underlying density distribution. For clustering studies, it may be useful to fit a mixture model with many components and to divide components into "clusters" and "background" by setting a density threshold; for an example of this approach see figures 10.20 and 10.21.

An additional important factor that influences the number of mixture components supported by data is the sample size. Figure 6.9 illustrates how the best-fit GMM changes dramatically as the sample size is increased from 100 to 1000. Furthermore, even when the sample includes as many as 10,000 points, the underlying model is not fully recovered (only one of the two background components is recognized).

6.3.2. Cloning Data in $D > 1$ Dimensions

Here we return briefly to a subject we discussed in §3.7: cloning a distribution of data. The rank-based approach illustrated in figure 3.25 works well in one dimension, but cloning an arbitrary higher-dimensional distribution requires an estimate of the local density at each point. Gaussian mixtures are a natural choice for this, because they can flexibly model density fields in any number of dimensions, and easily generate new points within the model.

Figure 6.10 shows the procedure: from 1000 observed points, we fit a ten-component Gaussian mixture model to the density. A sample of 5000 points drawn from this density model mimics the input to the extent that the density model is accurate. This idea can be very useful when simulating large multidimensional data sets based on small observed samples. This idea will also become important in the following section, in which we explore a variant of Gaussian mixtures in order to create denoised samples from density models based on noisy observed data sets.

6.3.3. GMM with Errors: Extreme Deconvolution

Bayesian estimation of multivariate densities modeled as mixtures of Gaussians, with data that have measurement error, is known in astronomy as "extreme deconvolution" (XD); see [6]. As with the Gaussian mixtures above, we have already encountered this situation in one dimension in §4.4. Recall the original mixture of Gaussians, where each data point $\mathbf{x}$ is sampled from one of M different Gaussians with given means and variances, $(\mu_i, \mathbf{\Sigma}_i)$, with the weight for each Gaussian being

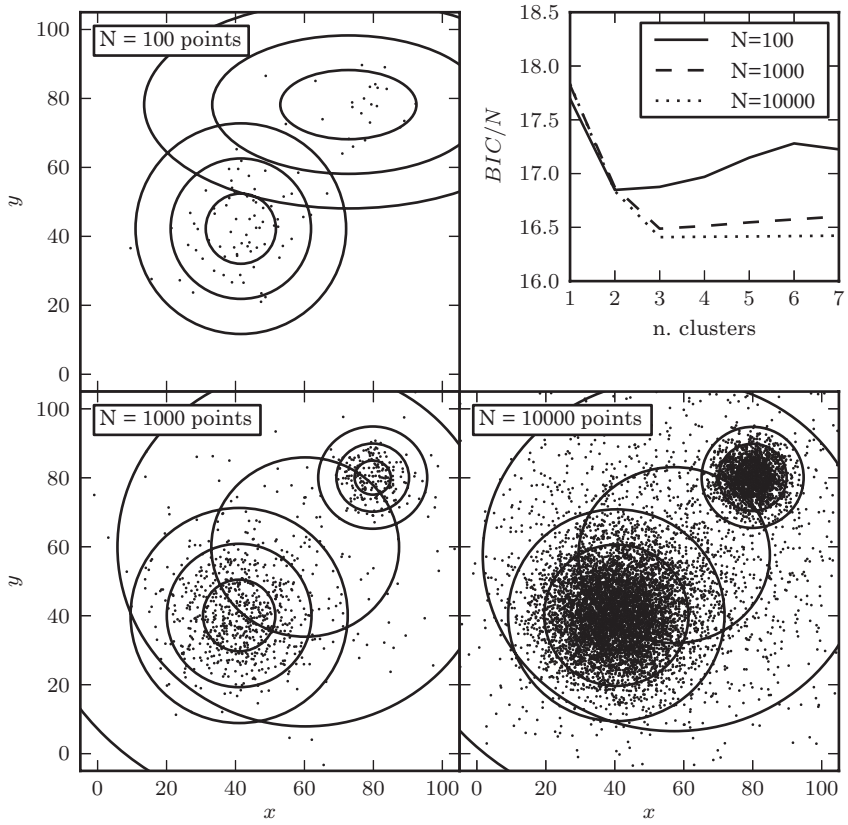


Figure 6.9. The BIC-optimized number of components in a Gaussian mixture model as a function of the sample size. All three samples (with 100, 1000, and 10,000 points) are drawn from the same distribution: two narrow foreground Gaussians and two wide background Gaussians. The top-right panel shows the BIC as a function of the number of components in the mixture. The remaining panels show the distribution of points in the sample and the 1, 2, and 3 standard deviation contours of the best-fit mixture model.

α_i. Thus, the pdf of $\mathbf{x}$ is given as

$$p(\mathbf{x}) = \sum_j \alpha_j \mathcal{N}(\mathbf{x}|\mu_j, \mathbf{\Sigma}_j)\,, \tag{6.18}$$

where, recalling eq. 3.97,

$$\mathcal{N}(\mathbf{x}|\mu_j, \mathbf{\Sigma}_j) = \frac{1}{\sqrt{(2\pi)^D \det(\mathbf{\Sigma}_j)}} \exp\Big(-\frac{1}{2}(\mathbf{x}-\mu)^T \mathbf{\Sigma}_j^{-1}(\mathbf{x}-\mu)\Big)\,. \tag{6.19}$$

Extreme deconvolution generalizes the EM approach to a case with measurement errors. More explicitly, one assumes that the noisy observations $\mathbf{x}_i$ and the true values $\mathbf{v}_i$ are related through

$$\mathbf{x}_i = \mathbf{R}_i \mathbf{v}_i + \epsilon_i, \tag{6.20}$$

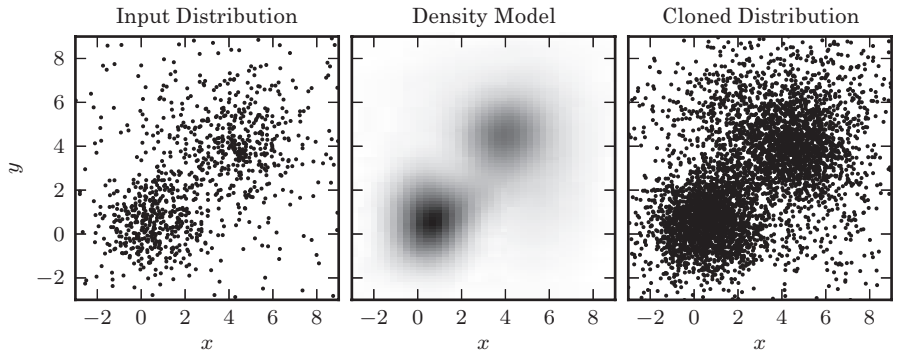


Figure 6.10. Cloning a two-dimensional distribution. The left panel shows 1000 observed points. The center panel shows a ten-component Gaussian mixture model fit to the data (two components dominate over other eight). The third panel shows 5000 points drawn from the model in the second panel.

where $\mathbf{R}_i$ is the so-called projection matrix, which may or may not be invertible. The noise ϵ_i is assumed to be drawn from a Gaussian with zero mean and variance $\mathbf{S}_i$. Given the matrices $\mathbf{R}_i$ and $\mathbf{S}_i$, the aim of XD is to find the parameters μ_i, $\mathbf{\Sigma}_i$ of the underlying Gaussians, and the weights α_i, as defined in §6.18, in a way that would maximize the likelihood of the observed data. The EM approach to this problem results in an iterative procedure that converges to (at least) a local maximum of the likelihood. The generalization of the EM procedure (see [6]) in §4.4.3 becomes the following:

- The expectation (E) step:

$$q_{ij} \leftarrow \frac{\alpha_j \mathcal{N}(\mathbf{w}_i | \mathbf{R}_i \mu_j, \mathbf{T}_{ij})}{\sum_j \alpha_k \mathcal{N}(\mathbf{w}_i | \mathbf{R}_i \mu_k, \mathbf{T}_{ij})}, \tag{6.21}$$

$$\mathbf{b}_{ij} \leftarrow \mu_j + \mathbf{\Sigma}_j \mathbf{R}_i^T \mathbf{T}_{ij}^{-1} (\mathbf{w}_i - \mathbf{R}_i \mu_j), \tag{6.22}$$

$$\mathbf{B}_{ij} \leftarrow \mathbf{\Sigma}_j - \mathbf{\Sigma}_j \mathbf{R}_i^T \mathbf{T}_{ij}^{-1} \mathbf{R}_i \mathbf{\Sigma}_j, \tag{6.23}$$

  where $\mathbf{T}_{ij} = \mathbf{R}_i \mathbf{\Sigma}_j \mathbf{R}_i^T + \mathbf{S}_i$.
- The maximization (M) step:

$$\alpha_i \leftarrow \frac{1}{N} \sum_i q_{ij}, \tag{6.24}$$

$$\mu_j \leftarrow \frac{1}{q_j} \sum_i q_{ij} \mathbf{b}_{ij}, \tag{6.25}$$

$$\mathbf{\Sigma}_j \leftarrow \frac{1}{q_j} \sum_i q_{ij} [(\mu_j - \mathbf{b}_{ij})(\mu_j - \mathbf{b}_{ij}^T) + \mathbf{B}_{ij}], \tag{6.26}$$

  where $q_j = \sum_i q_{ij}$.

The iteration of these steps increases the likelihood of the observations $\mathbf{w}_i$, given the model parameters. Thus, iterating until convergence, one obtains a solution that is a local maximum of the likelihood. This method has been used with success in quasar classification, by estimating the densities of quasar and nonquasar objects from flux measurements; see [5]. Details of the use of XD, including methods to avoid local maxima in the likelihood surface, can be found in [6].

AstroML contains an implementation of XD which has a similar interface to `GMM` in Scikit-learn:

```
import numpy as np
from astroML.density_estimation import XDGMM

X = np.random.normal(size=(1000, 1)) # 1000 pts in
    # 1 dim
Xerr = np.random.random((1000, 1, 1)) # 1000 1x1
        # covariance matrices
xdgmm = XDGMM(n_components=2)
xdgmm.fit(X, Xerr) # fit the model
logp=xdgmm.logprob_a(X, Xerr) # evaluate probability
X_new =  xdgmm.sample(1000) # sample new points from
         #  distribution
```

For further examples, see the source code of figures 6.11 and 6.12.

Figure 6.11 shows the performance of XD on a simulated data set. The top panels show the true data set (2000 points) and the data set with noise added. The bottom panels show the XD results: on the left is a new data set drawn from the mixture (as expected, it has the same characteristics as the noiseless sample). On the right are the 2σ limits of the ten Gaussians used in the fit. The important feature of this figure is that from the noisy data, we are able to recover a distribution that closely matches the true underlying data: we have deconvolved the data and the noise in a similar vein to deconvolution KDE in §6.1.2.

This deconvolution of measurement errors can also be demonstrated using a real data set. Figure 6.12 shows the results of XD when applied to photometric data from the Sloan Digital Sky Survey. The high signal-to-noise data (i.e., small color errors; top-left panel) come from the Stripe 82 Standard Star Catalog, where multiple observations are averaged to arrive at magnitudes with a smaller scatter (via the central limit theorem; see §3.4). The lower signal-to-noise data (top-right panel) are derived from single epoch observations. Though only two dimensions are plotted, the XD fit is performed on a five-dimensional data set, consisting of the g-band magnitude along with the $u-g$, $g-r$, $r-i$, and $i-z$ colors.

The results of the XD fit to the noisy data are shown in the two middle panels: the background distribution is fit by a single wide Gaussian, while the remaining clusters trace the main locus of points. The points drawn from the resulting distribution have a much tighter scatter than the input data. This decreased scatter can be

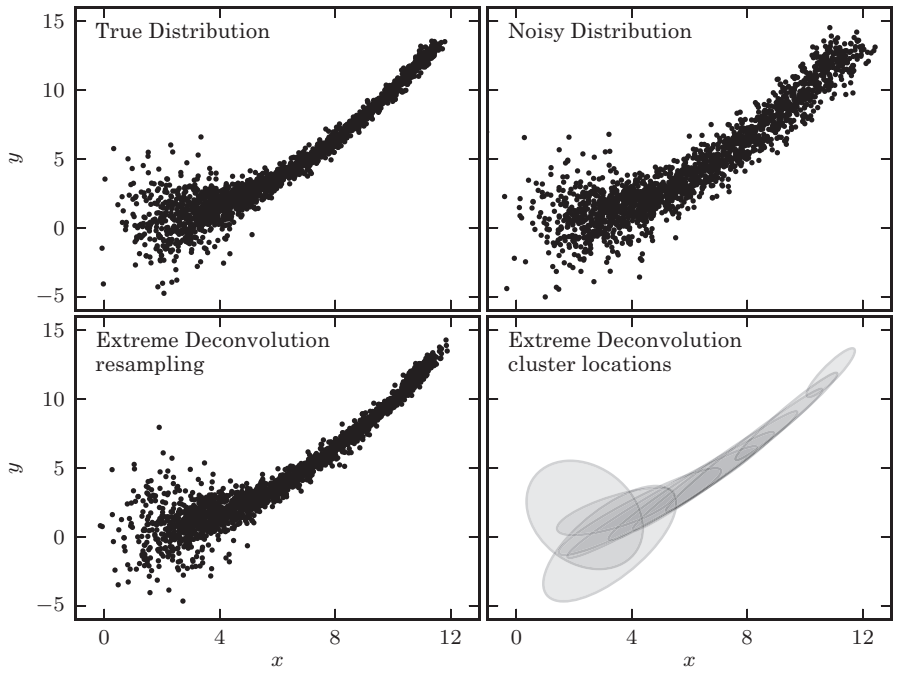


Figure 6.11. An example of extreme deconvolution showing a simulated two-dimensional distribution of points, where the positions are subject to errors. The top two panels show the distributions with small (left) and large (right) errors. The bottom panels show the densities derived from the noisy sample (top-right panel) using extreme deconvolution; the resulting distribution closely matches that shown in the top-left panel.

quantitatively demonstrated by analyzing the width of the locus perpendicular to its long direction using the so-called w color; see [17].

The w color is defined as

$$w = -0.227g + 0.792r - 0.567i + 0.05, \tag{6.27}$$

and has a zero mean by definition. The lower panel of figure 6.12 shows a histogram of the width of the w color in the range $0.3 < g-r < 1.0$ (i.e., along the "blue" part of the locus where w has a small standard deviation). The noisy data show a spread in w of 0.016 (magnitude), while the extreme deconvolution model reduces this to 0.008, better reflective of the true underlying distribution. Note that the intrinsic width of the w color obtained by XD is actually a bit smaller than the corresponding width for the Standard Star Catalog (0.010) because even the averaged data have residual random errors. By subtracting 0.008 from 0.010 in quadrature, we can estimate these errors to be 0.006, in agreement with independent estimates; see [17].

Last but not least, XD can gracefully treat cases of missing data: in this case the corresponding measurement error can be simply set to a very large value (much larger than the dynamic range spanned by available data).

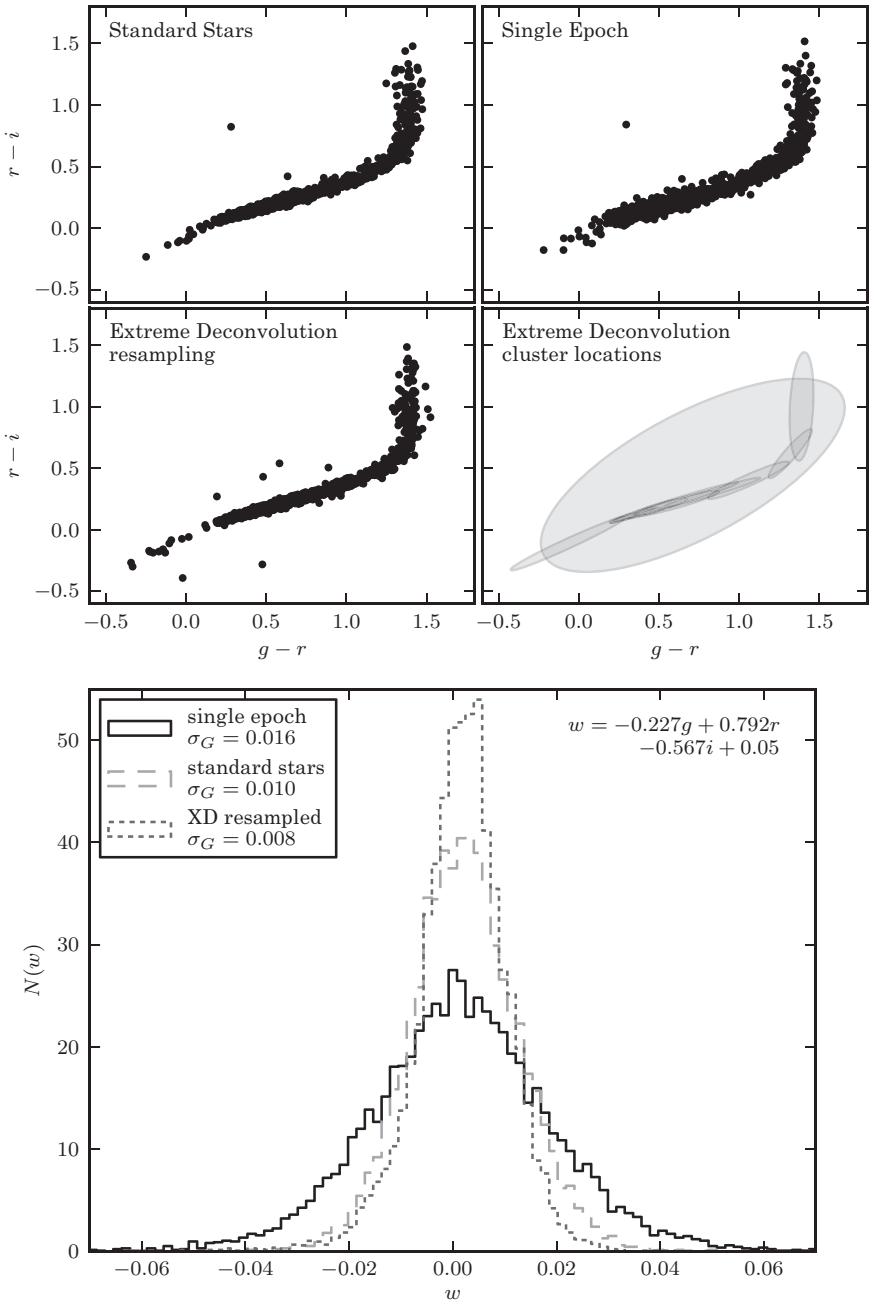


Figure 6.12. Extreme deconvolution applied to stellar data from SDSS Stripe 82. The top panels compare the color distributions for a high signal-to-noise sample of standard stars (left) with lower signal-to-noise, single epoch, data (right). The middle panels show the results of applying extreme deconvolution to the single epoch data. The bottom panel compares the distributions of a color measured perpendicularly to the locus (the so-called w color is defined following [16]). The distribution of colors from the extreme deconvolution of the noisy data recovers the tight distribution of the high signal-to-noise data.

6.4. Finding Clusters in Data

"Clustering" in astronomy refers to a number of different aspects of data analysis. Given a multivariate point data set, we can ask whether it displays any structure, that is, concentrations of points. Alternatively, when a density estimate is available we can search for "overdensities." Another way to interpret clustering is to seek a partitioning or segmentation of data into smaller parts according to some criteria. In the following section we describe the techniques used for the unsupervised identification of clusters within point data sets. Again, here "unsupervised" means that there is no prior information about the number and properties of clusters.

6.4.1. General Aspects of Clustering and Unsupervised Learning

Finding clusters is sometimes thought of as "black art" since the objective criteria for it seems more elusive than, say, for a prediction task such as classification (where we know the true underlying function for at least some subset of the sample). When we can speak of a *true underlying function* (as we do in most density estimation, classification, and regression methods) we mean that we have a score or error function with which to evaluate the effectiveness of our analysis. Under this model we can discuss optimization, error bounds, generalization (i.e., minimizing error on future data), what happens to the error as we get more data, etc. In other words we can leverage all the powerful tools of statistics we have discussed previously.

6.4.2. Clustering by Sum-of-Squares Minimization: K-Means

One of the simplest methods for partitioning data into a small number of clusters is K-means. K-means seeks a partitioning of the points into K disjoint subsets C_k with each subset containing N_k points such that the following sum-of-squares objective function is minimized:

$$\sum_{k=1}^{K} \sum_{i \in C_k} ||x_i - \mu_k||^2, \tag{6.28}$$

where $\mu_k = \frac{1}{N_k} \sum_{i \in C_k} x_i$ is the mean of the points in set C_k, and $C(x_i) = C_k$ denotes that the class of x_i is C_k.

The procedure for K-means is to initially choose the centroid, μ_k, of each of the K clusters. We then assign each point to the cluster that it is closest to (i.e., according to $C(x_i) = \arg\min_k ||x_i - \mu_k||$). At this point we update the centroid of each cluster by recomputing μ_k according to the new assignments. The process continues until there are no new assignments.

While a globally optimal minimum cannot be guaranteed, the process can be shown to never increase the sum-of-squares error. In practice K-means is run multiple times with different starting values for the centroids of C_k and the result with the lowest sum-of-squares error is used. K-means can be interpreted as a "hard" version of the EM algorithm for a mixture of spherical Gaussians (i.e., we are assuming with K-means that the data can be described by spherical clusters with each cluster containing approximately the same number of points).

Scikit-learn implements K-means using expectation maximization:

```
import numpy as np
from sklearn.cluster import KMeans

X = np.random.normal(size=(1000, 2)) # 1000 pts in 2
    #dims
clf = KMeans(n_clusters=3)
clf.fit(X)
centers=clf.cluster_centers_ # locations of the
        # clusters
labels=clf.predict(X) # labels for each of the
       # points
```

For more information, see the Scikit-learn documentation.

In figure 6.13 we show the application of K-means to the stellar metallicity data used for the Gaussian mixture model example in figure 6.6. For the $K = 4$ clusters (consistent with figure 6.6) we find that the background distribution "pulls" the centroid of two of the clusters such that they are offset from the peak of the density distributions. This contrasts with the results found for the GMM described in §6.3 where we model the two density peaks (with the additional Gaussians capturing the structure in the distribution of background points).

6.4.3. Clustering by Max-Radius Minimization: the Gonzalez Algorithm

An alternative to minimizing the sum of square errors is to minimize the maximum radius of a cluster,

$$\min_k \max_{x_i \in C_k} ||x_i - \mu_k||, \tag{6.29}$$

where we assign one of the points within the data set to be the center of each cluster, μ_k.

An effective algorithm for finding the cluster centroids is known as the *Gonzalez algorithm*. Starting with no clusters we progressively add one cluster at a time (by arbitrarily selecting a point within the data set to be the center of the cluster). We then find the point x_i which maximizes the distance from the centers of existing clusters and set that as the next cluster center. This procedure is repeated until we achieve K clusters. At this stage each point in the data set is assigned the label of its nearest cluster center.

6.4.4. Clustering by Nonparametric Density Estimation: Mean Shift

Another way to find arbitrarily shaped clusters is to define clusters in terms of the modes or peaks of the nonparametric density estimate, associating each data point with its closest peak. This so-called *mean-shift* algorithm is a technique to find local modes (bumps) in a kernel density estimate of the data. The concept behind mean

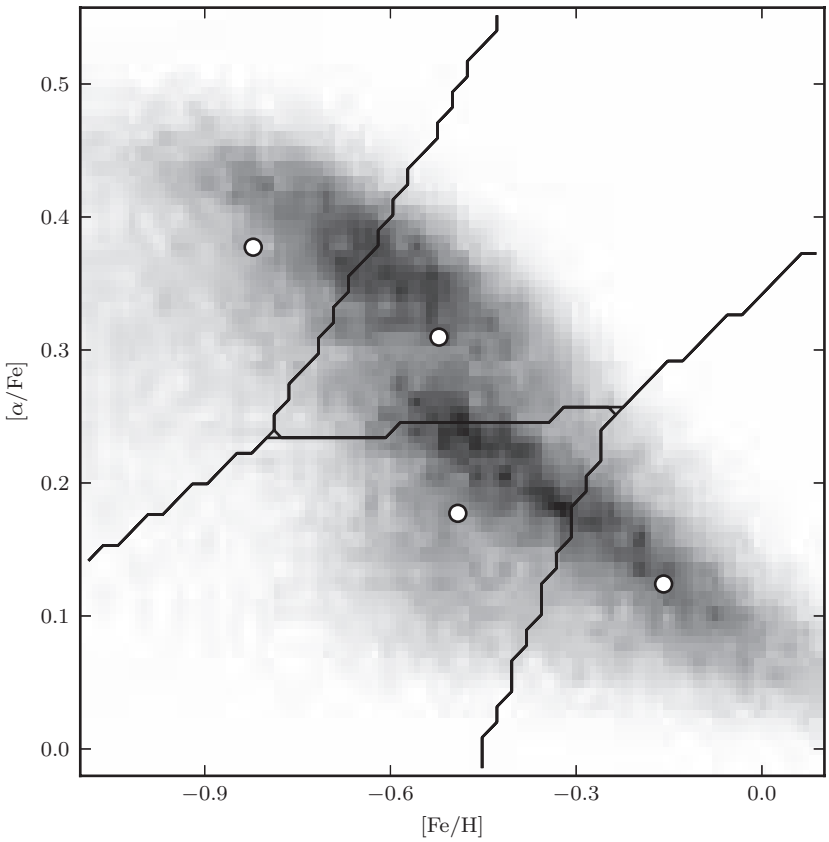


Figure 6.13. The K-means analysis of the stellar metallicity data used in figure 6.6. Note how the background distribution "pulls" the cluster centers away from the locus where one would place them by eye. This is why more sophisticated models like GMM are often better in practice.

shift is that we move the data points in the direction of the log of the gradient of the density of the data, until they finally converge to each other at the peaks of the bumps. The number of modes, K, is found implicitly by the method.

Suppose x_i^m is the position of the ith data point during iteration m of the procedure. A kernel density estimate $\widehat{f}^m$ is constructed from the points $\{x_i^m\}$. We obtain the next round of points according to an update procedure:

$$x_i^{m+1} = x_i^m + a\nabla \log \widehat{f}^m(x_i^m) \tag{6.30}$$

$$= x_i^m + \frac{a}{\widehat{f}^m(x_i^m)}\nabla \widehat{f}^m(x_i^m), \tag{6.31}$$

where $\widehat{f}^m(x_i^m)$ is found by kernel density estimation and $\nabla \widehat{f}^m(x_i^m)$ is found by kernel density estimation using the gradient of the original kernel.

The convergence of this procedure is defined by the bandwidth, h, of the kernel and the parametrization of a. For example, points drawn from a spherical Gaussian will jump to the centroid in one step when a is set to the variance of the Gaussian. The log of the gradient of the density ensures that the method converges in a few iterations, with points in regions of low density moving a considerable distance toward regions of high density in each iteration.

For the Epanechnikov kernel (see §6.1.1) and the value

$$a = \frac{h^2}{D+2}, \tag{6.32}$$

the update rule reduces to the form

$$x_i^{m+1} = \text{mean position of points } x_i^m \text{ within distance } h \text{ of } x_i^m. \tag{6.33}$$

This is called the *mean-shift algorithm*.

Mean shift is implemented in Scikit-learn:

```
import numpy as np
from sklearn.cluster import MeanShift

X = np.random.normal(size=(1000, 2)) # 1000 pts in 2
    # dims
ms = MeanShift(bandwidth=1.0)
     # if no bandwidth is specified,
     # it will be learned from data
ms.fit(X) # fit the data
centers = ms.cluster_centers_ # centers of clusters
labels = ms.labels_ # labels of each point X
```

For more examples and information, see the source code for figure 6.14 and the Scikit-learn documentation.

An example of the mean-shift algorithm is shown in figure 6.14 using the same metallicity data set used in figures 6.6 and 6.13. The algorithm identifies the modes (or bumps) within the density distributions without attempting to model the correlation of the data within the clusters (i.e., the resulting clusters are axis aligned).

6.4.5. Clustering Procedurally: Hierarchical Clustering

A *procedural* method is a method which has not been formally related to some function of the underlying density. Such methods are more common in clustering

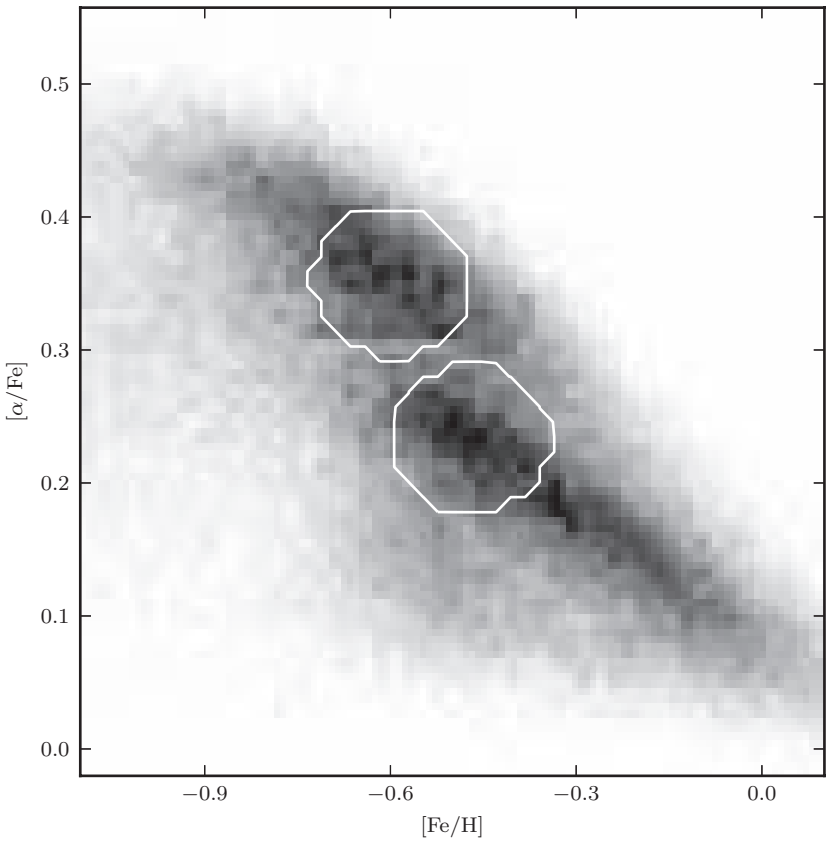


Figure 6.14. Mean-shift clustering on the metallicity data set used in figures 6.6 and 6.13. The method finds two clusters associated with local maxima of the distribution (interior of the circles). Points outside the circles have been determined to lie in the background. The mean shift does not attempt to model correlation in the clusters: that is, the resulting clusters are axis aligned.

and dimension reduction than in other tasks. Although this makes it hard, if not impossible, to say much about these methods analytically, they are nonetheless often still useful in practice.

Hierarchical clustering relaxes the need to specify the number of clusters K by finding all clusters at all scales. We start by partitioning the data into N clusters, one for each point in the data set. We can then join two of the clusters resulting in $N - 1$ clusters. This procedure is repeated until the Nth partition contains one cluster. If two points are in the same cluster at level m, and remain together at all subsequent levels, this is known as *hierarchical clustering* and is visualized using a tree diagram or *dendrogram*.

Hierarchical clustering can be approached as a top-down (*divisive*) procedure, where we progressively subdivide the data, or as a bottom-up (*agglomerative*) procedure, where we merge the nearest pairs of clusters. For our examples below we will consider the agglomerative approach.

At each step in the clustering process we merge the "nearest" pair of clusters. Options for defining the distance between two clusters, C_k and $C_{k'}$, include

$$d_{\min}(C_k, C_{k'}) = \min_{x \in C_k, x' \in C_{k'}} ||x - x'||, \tag{6.34}$$

$$d_{\max}(C_k, C_{k'}) = \max_{x \in C_k, x' \in C_{k'}} ||x - x'||, \tag{6.35}$$

$$d_{\text{avg}}(C_k, C_{k'}) = \frac{1}{N_k N_{k'}} \sum_{x \in C_k} \sum_{x' \in C_{k'}} ||x - x'||, \tag{6.36}$$

$$d_{\text{cen}}(C_k, C_{k'}) = ||\mu_k - \mu_{k'}||, \tag{6.37}$$

where x and x' are the points in cluster C_k and $C_{k'}$ respectively, N_k and $N_{k'}$ are the number of points in each cluster, and μ_k and $\mu_{k'}$ the centroid of the clusters.

Using the distance $d_{\min}$ results in a hierarchical clustering known as a minimum spanning tree (see [1, 4, 20], for some astronomical applications) and will commonly produce clusters with extended chains of points. Using $d_{\max}$ tends to produce hierarchical clustering with compact clusters. The other two distance examples have behavior somewhere between these two extremes.

A hierarchical clustering model, using $d_{\min}$, for the SDSS "Great Wall" data is shown in figure 6.15. The extended chains of points expressed by this minimum spanning tree trace the large-scale structure present within the data. Individual clusters can be isolated by sorting the links (or edges as they are known in graph theory) by increasing length, then deleting those edges longer than some threshold. The remaining components form the clusters. For a *single-linkage hierarchical clustering* this is also known as "friends-of-friends" clustering [31], and in astronomy is often used in cluster analysis for N-body simulations (e.g., [2, 9]).

Unfortunately a minimum spanning tree is naively $\mathcal{O}(N^3)$ to compute, using straightforward algorithms. Well-known graph-based algorithms, such as Kruskal's [19] and Prim's [32] algorithms, must consider the entire set of edges (see WSAS for details). In our Euclidean setting, there are $\mathcal{O}(N^2)$ possible edges, rendering these algorithms too slow for large data sets. However it has recently been shown how to perform the computation in approximately $\mathcal{O}(N \log N)$ time; see [27].

Figure 6.15 shows an *approximate* Euclidean minimum spanning tree, which finds the minimum spanning tree of the graph built using the k nearest neighbors of each point. The calculation is enabled by utility functions in SciPy and Scikit-learn:

```
from scipy.sparse.csgraph import\
     minimum_spanning_tree
from sklearn.neighbors import kneighbors_graph

X = np.random.random((1000, 2)) # 1000 pts in 2 dims
G = kneighbors_graph(X, n_neighbors=10,
                     mode='distance')
T = minimum_spanning_tree(G)
```

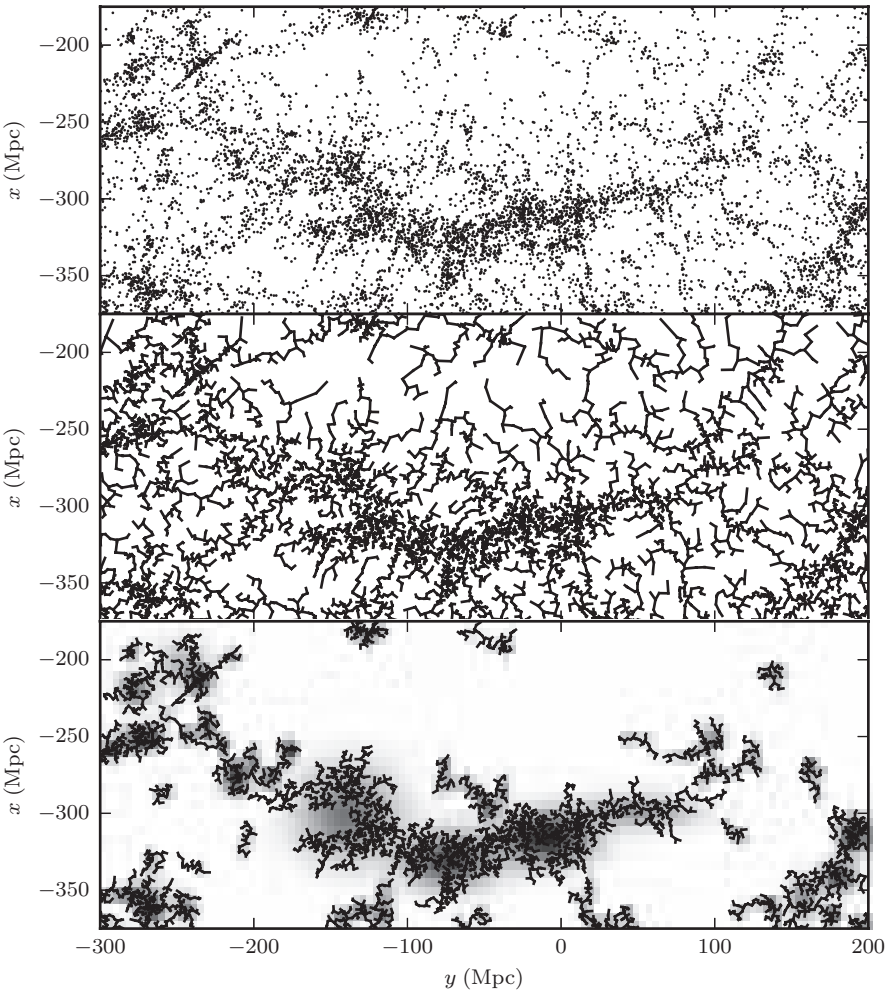


Figure 6.15. An approximate Euclidean minimum spanning tree over the two-dimensional projection of the SDSS Great Wall. The upper panel shows the input points, and the middle panel shows the dendrogram connecting them. The lower panel shows clustering based on this dendrogram, created by removing the largest 10% of the graph edges, and keeping the remaining connected clusters with 30 or more members. See color plate 4.

The result is that `T` is a 1000 × 1000 sparse matrix, with `T[i, j] = 0` for nonconnected points, and `T[i, j]` is the distance between points `i` and `j` for connected points. This algorithm will be efficient for small `n_neighbors`. If we set `n_neighbors=1000` in this case, then the approximation to the Euclidean minimum spanning tree will be exact. For well-behaved data sets, this approximation is often exact even for $k \ll N$. For more details, see the source code of figure 6.15.

6.5. Correlation Functions

In earlier sections we described the search for structure within point data using density estimation (§6.1) and cluster identification (§6.4). For point processes, a popular extension to these ideas is the use of correlation functions to characterize how far (and on what scales) the distribution of points differs from a random distribution; see [28, 30]. Correlation functions, and in particular autocorrelation functions, have been used extensively throughout astrophysics with examples of their use including the characterization of the fluctuations in the densities of galaxies and quasars as a function of luminosity, galaxy type and age of the universe. The key aspect of these statistics is that they can be used as metrics for testing models of structure formation and evolution directly against data.

We can define the correlation function by noting that the probability of finding a point in a volume element, dV, is directly proportional to the density of points, ρ. The probability of finding a pair of points in two volume elements, dV_1 and dV_2, separated by a distance, r, (see the left panel of figure 6.16) is then given by

$$dP_{12} = \rho^2 dV_1 dV_2 (1 + \xi(r)), \tag{6.38}$$

where $\xi(r)$ is known as the *two-point correlation function*.

From this definition, we see that the two-point correlation function describes the excess probability of finding a pair of points, as a function of separation, compared to a random distribution. Positive, negative, or zero amplitudes in $\xi(r)$ correspond to distributions that are respectively correlated, anticorrelated or random. The two-point correlation function relates directly to the power spectrum, $P(k)$, through the Fourier transform (see §10.2.2),

$$\xi(r) = \frac{1}{2\pi^2} \int dk\, k^2\, P(k)\, \frac{\sin(kr)}{kr} \tag{6.39}$$

with the scale or wavelength of a fluctuation, λ is related to the wave number k by $k = 2\pi/\lambda$. As such, the correlation function can be used to describe the density fluctuations of sources by

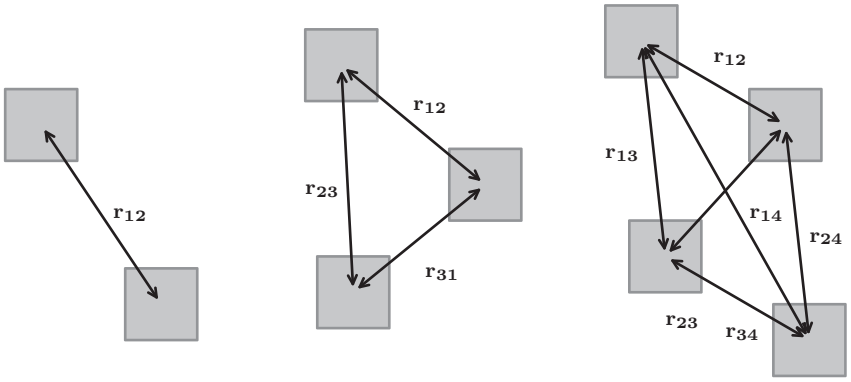
$$\xi(r) = \left\langle \frac{\delta\rho(x)}{\rho} \frac{\delta\rho(x+r)}{\rho} \right\rangle, \tag{6.40}$$

where $\delta\rho(x)/\rho = (\rho - \bar{\rho})/\rho$ is the density contrast, relative to the mean value $\bar{\rho}$, at position x.

In studies of galaxy distributions, $\xi(r)$ is often parametrized in terms of a power law,

$$\xi(r) = \left(\frac{r}{r_0}\right)^{-\gamma}, \tag{6.41}$$

where r_0 is the clustering scale length and γ the power law exponent (with $r_0 \sim 6\,\mathrm{Mpc}$ and $\gamma \sim 1.8$ for galaxies in the local universe). Rather than considering the full three-dimensional correlation function given by eq. 6.38, we often desire instead to look at the angular correlation function of the apparent positions of objects on the



(a) 2 point (b) 3 point (c) 4 point

Figure 6.16. An example of n-tuple configurations for the two-point, three-point, and four-point correlation functions (reproduced from WSAS).

sky (e.g., [8]). In this case, the approximate form of the relation is given by

$$w(\theta) = \left(\frac{\theta}{\theta_0}\right)^{\delta} \tag{6.42}$$

with $\delta = 1 - \gamma$ (see [26]).

Correlation functions can be extended to orders higher than the two-point function by considering configurations of points that comprise triplets (three-point function), quadruplets (four-point function), and higher multiplicities (n-point functions). Figure 6.16 shows examples of these configurations for the three- and four-point correlation function. Analogously to the definition of the two-point function, we can express these higher-order correlation functions in terms of the probability of finding a given configuration of points. For example, for the three-point correlation function we define the probability dP_{123} of finding three points in volume elements dV_1, dV_2, and dV_3 that are defined by a triangle with sides r_{12}, r_{13}, r_{23}. We write the three-point correlation function as

$$dP_{123} = \rho^3 dV_1 dV_2 dV_3 (1 + \xi(r_{12}) + \xi(r_{23}) + \xi(r_{13}) + \zeta(r_{12}, r_{23}, r_{13})) \tag{6.43}$$

with ζ known as the *reduced* or *connected three-point correlation function* (i.e., it does not depend on the lower-order correlation functions). The additional two-point correlation function terms in eq. 6.43 simply reflect triplets that arise from the excess of pairs of galaxies due to the nonrandom nature of the data.

6.5.1. Computing the n-point Correlation Function

n-point correlation functions are an example of the general n-point problems discussed in chapter 2. For simplicity, we start with the two-point correlation function, $\xi(r)$, which can be estimated by calculating the excess or deficit of pairs of points within a distance r and $r + dr$ compared to a random distribution. These random points are generated with the same selection function as the data (i.e., within the same volume and with identical masked regions)—the random data represent a Monte Carlo integration of the window function (see §10.2.2) of the data.

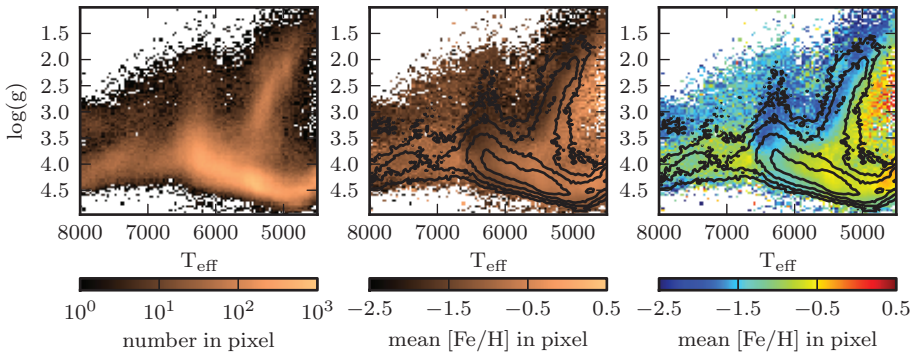


Plate 1. See figure 1.11.

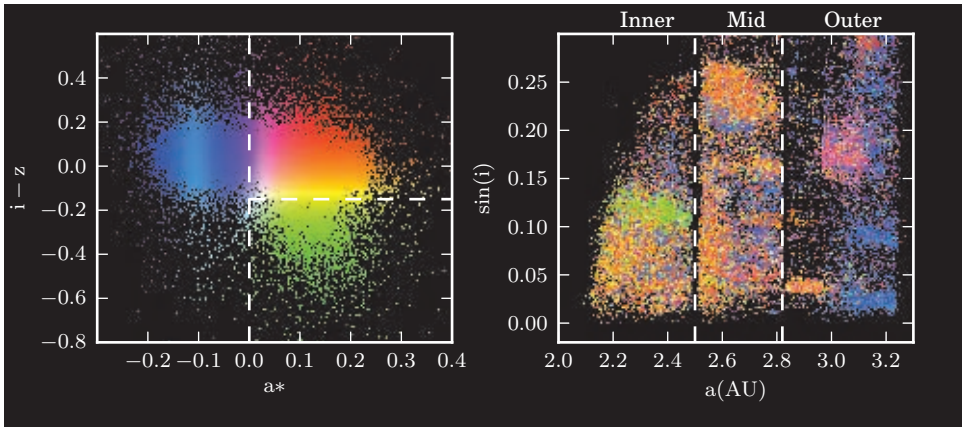


Plate 2. See figure 1.12.

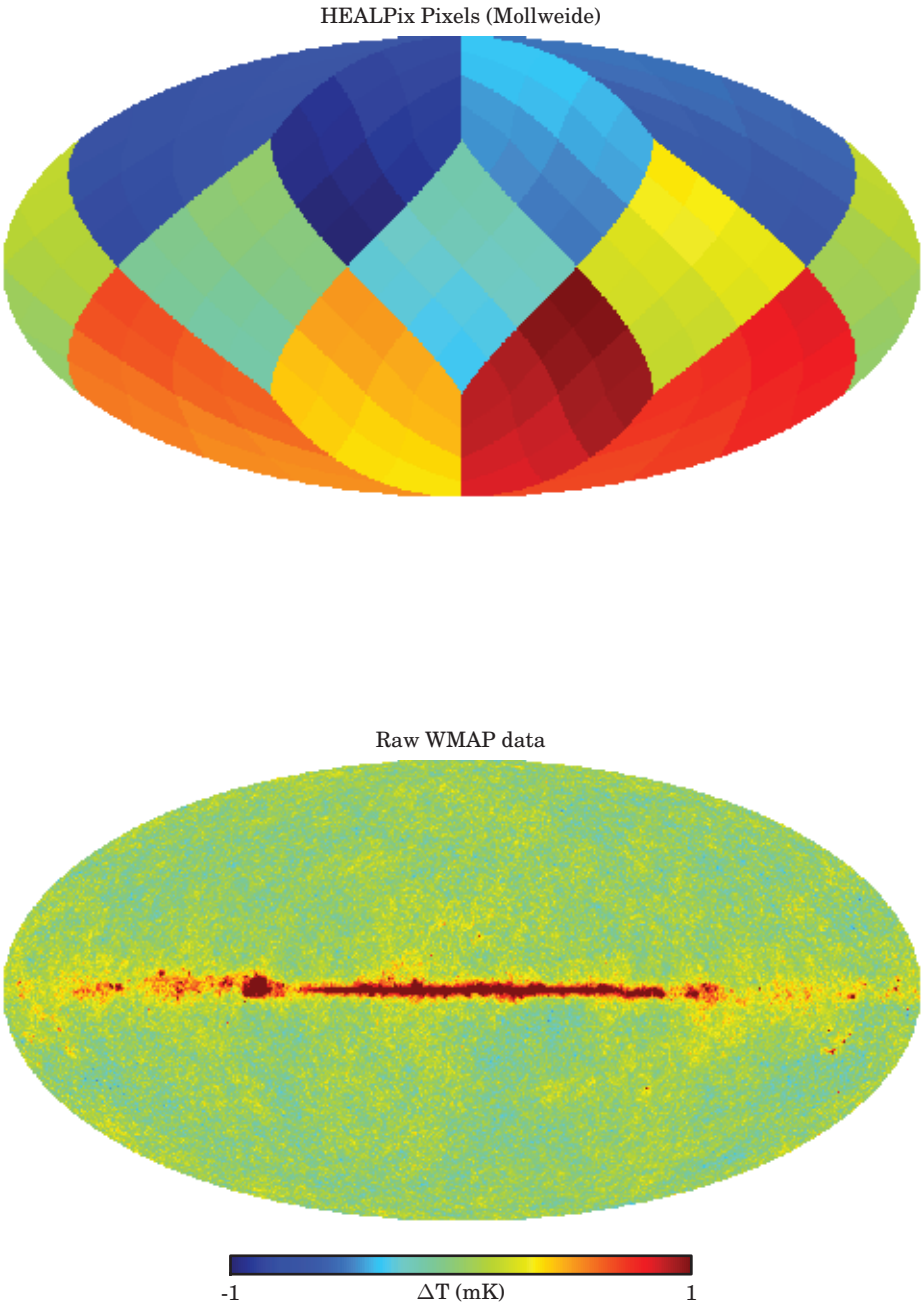


Plate 3. See figure 1.15.

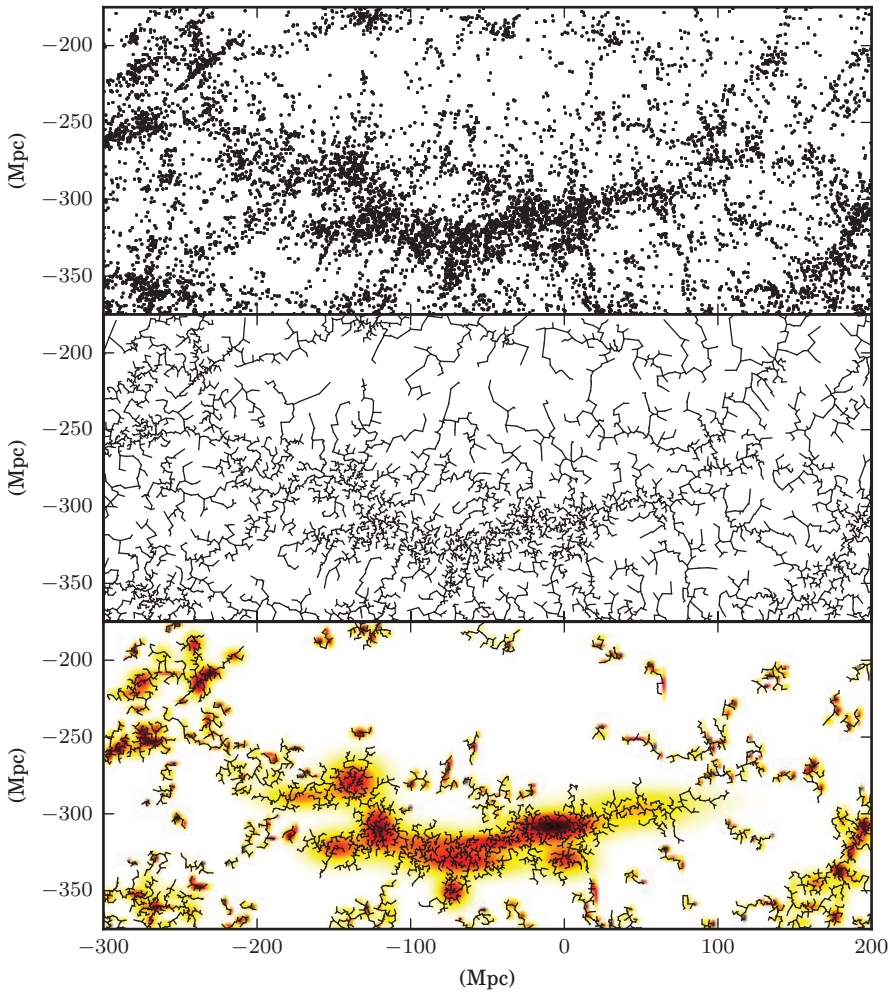


Plate 4. See figure 6.15.

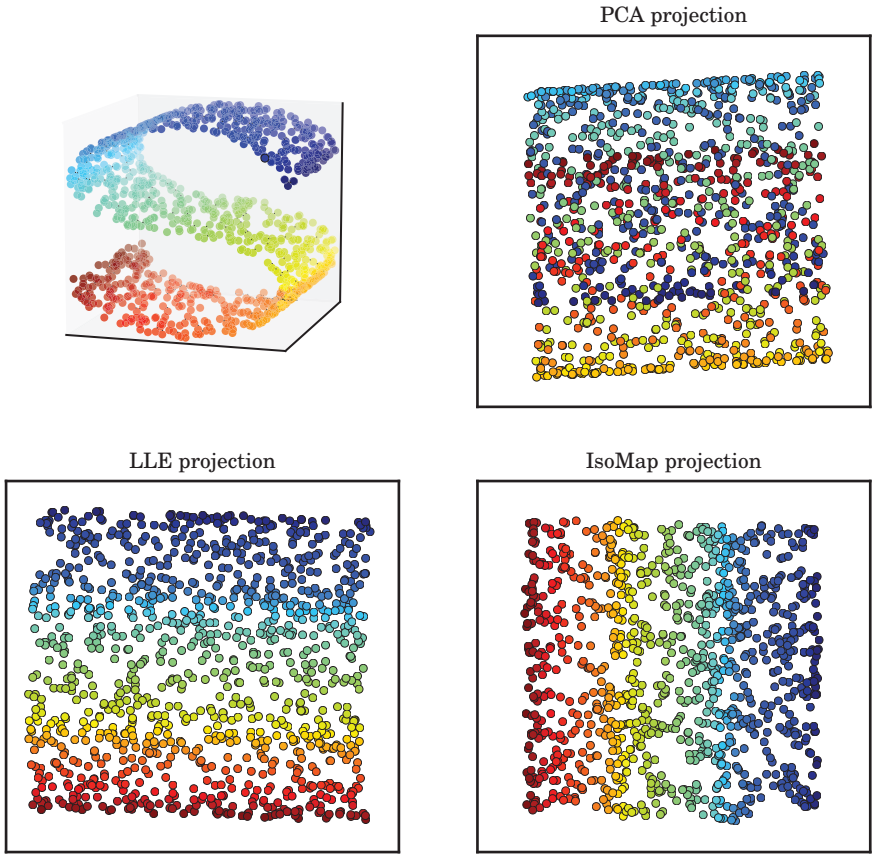


Plate 5. See figure 7.8.

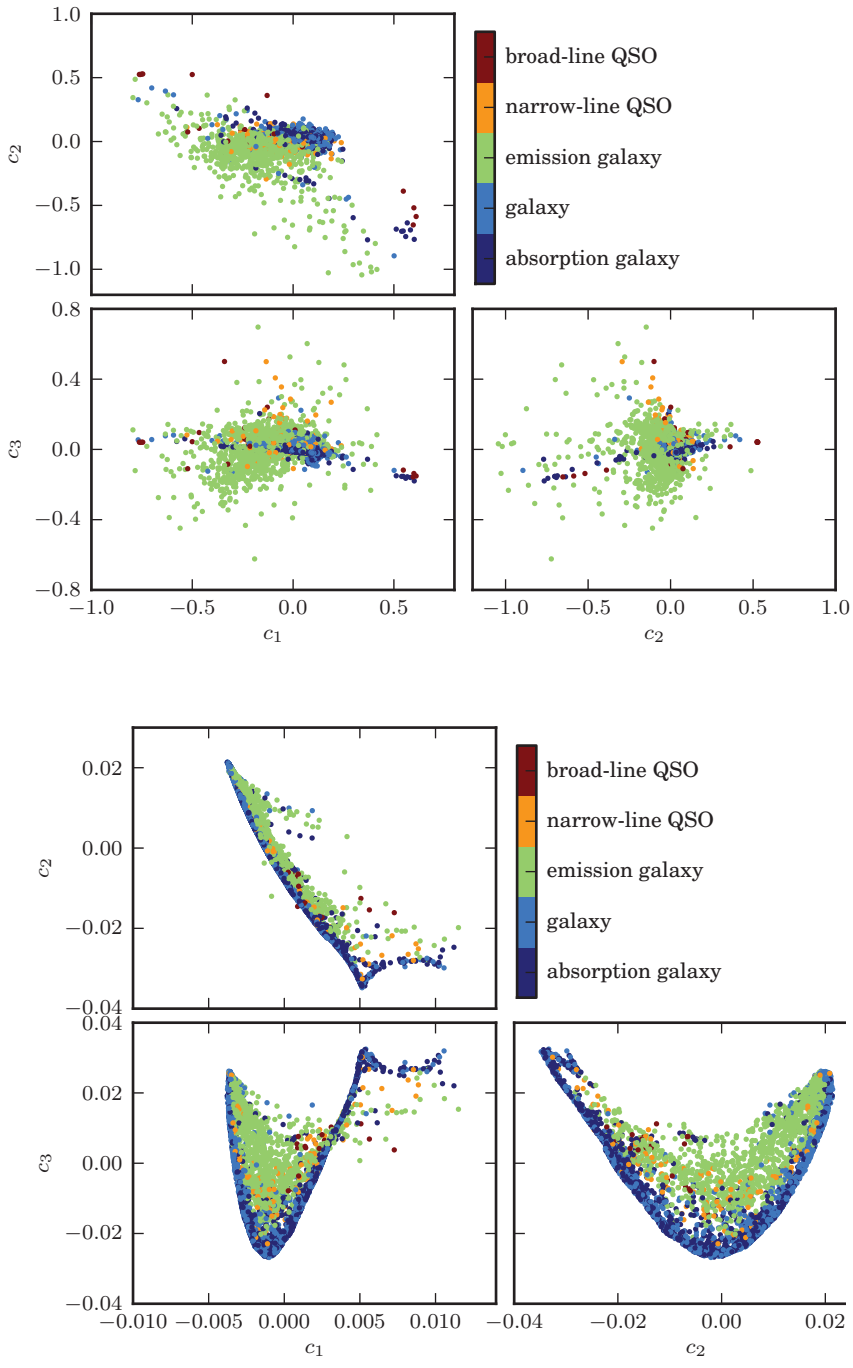


Plate 6. See figure 7.9.

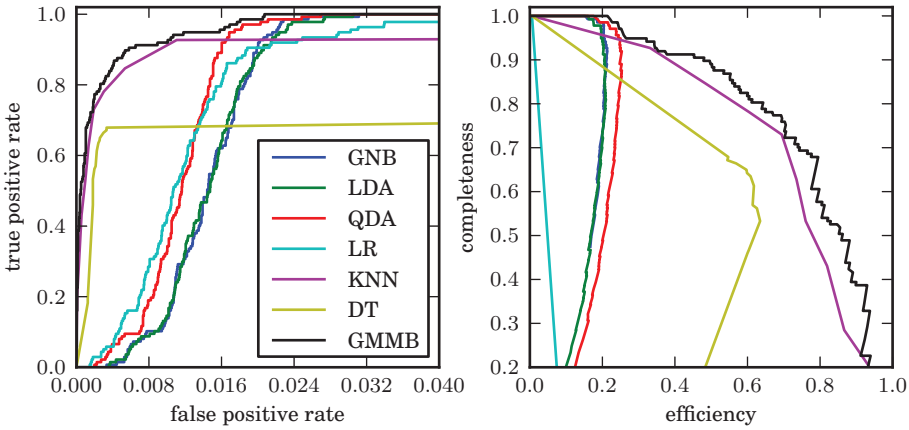


Plate 7. See figure 9.17.

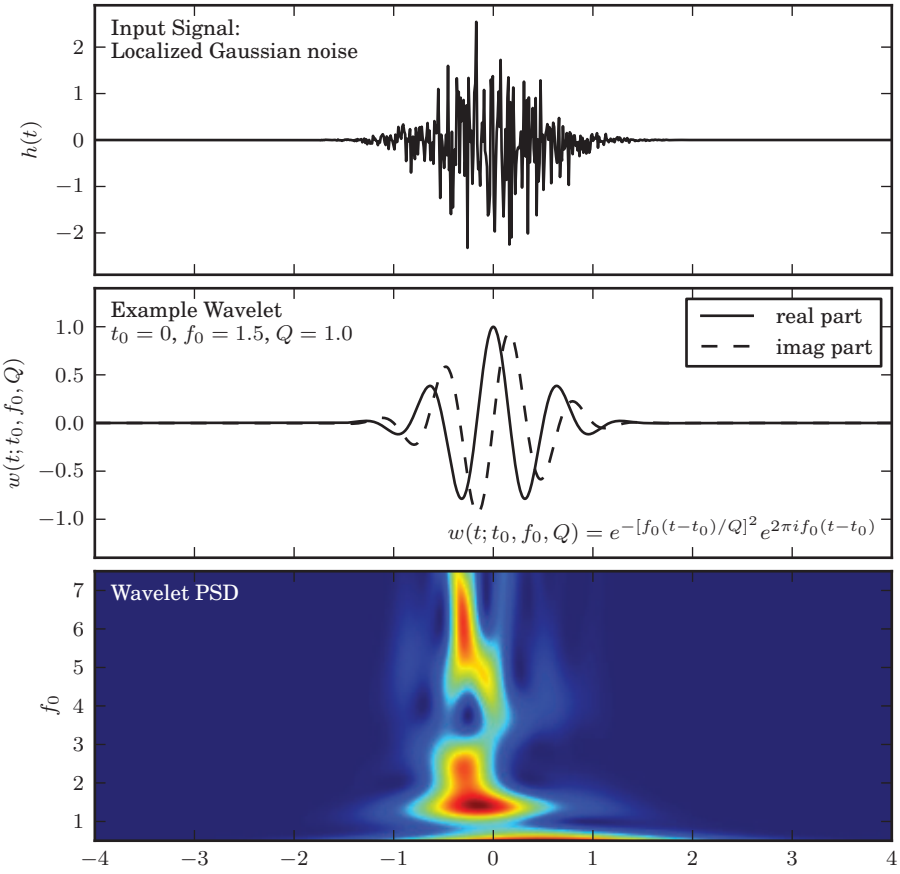


Plate 9. See figure 10.7.

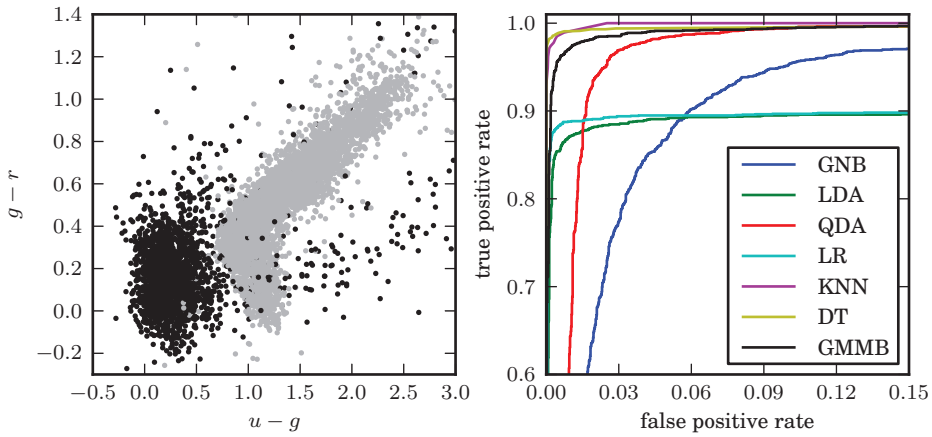


Plate 8. See figure 9.18.

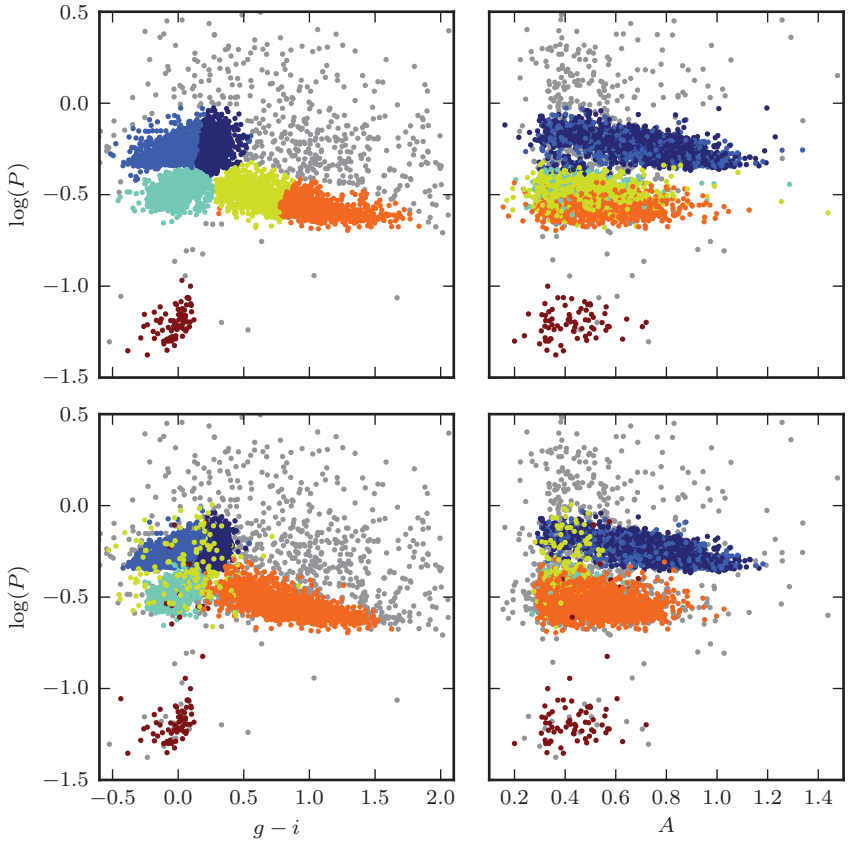


Plate 10. See figure 10.20.

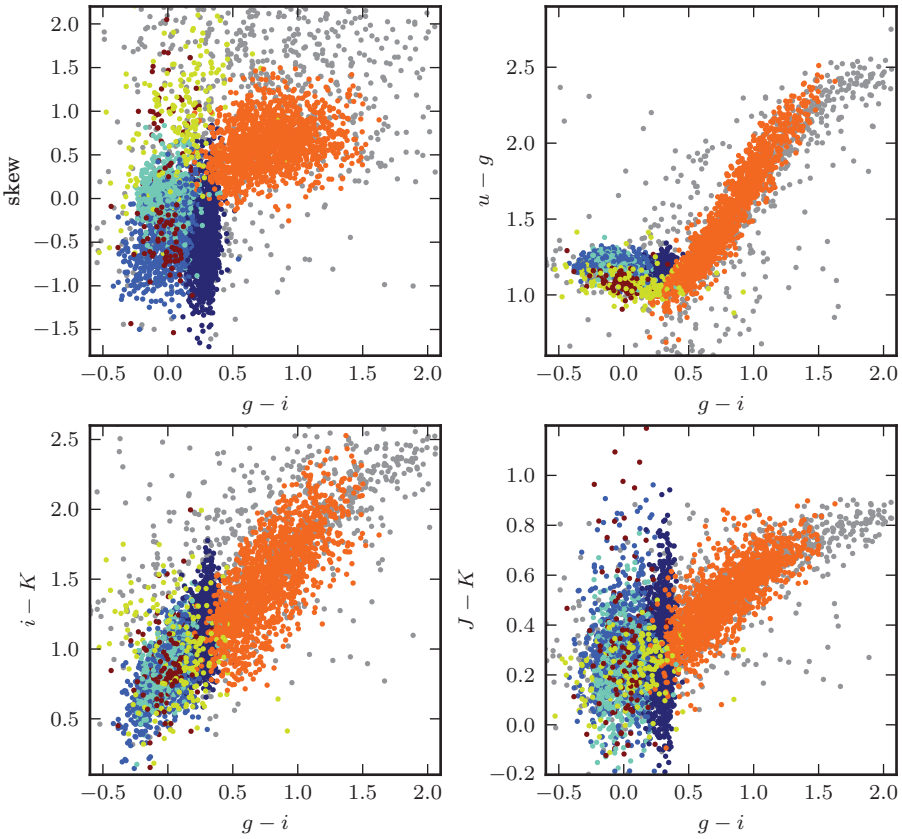


Plate 11. See figure 10.21.

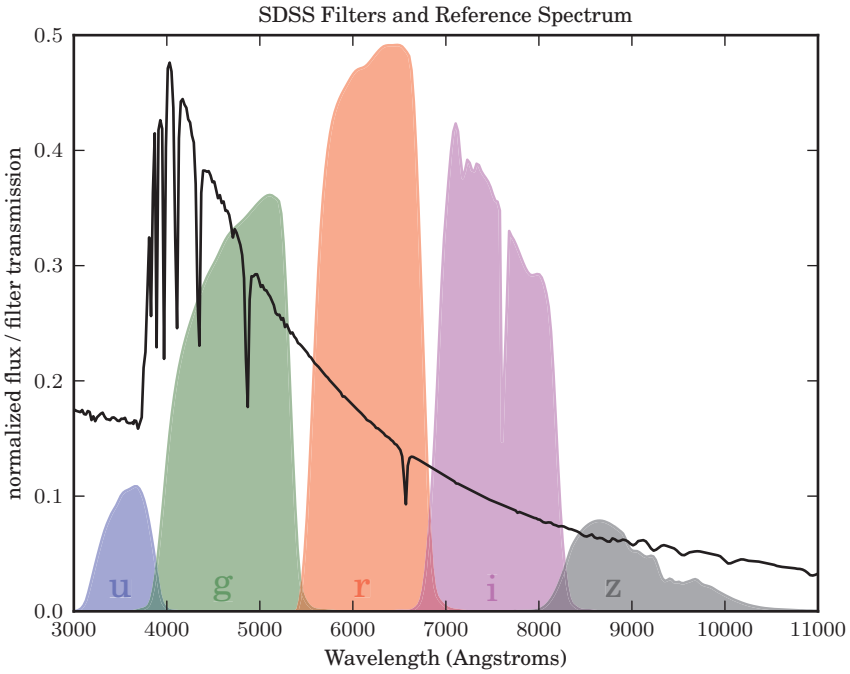


Plate 12. See figure C.1.

Typically the random data have a density ∼20 times higher than that of the data (to ensure that the shot noise of the randoms does not contribute to the variance of the estimator). This means that the computational cost of estimating the correlation function is dominated by the size of the random data set.

If we write the number of pairs of data points as $DD(r)$, the number of pairs of random points as $RR(r)$ and the number of data-random pairs as $DR(r)$, then we can write an estimator of the two-point correlation function as

$$\hat{\xi}(r) = \frac{DD(r)}{RR(r)} - 1. \tag{6.44}$$

Edge effects, due to the interaction between the distribution of sources and the irregular survey geometry in which the data reside, bias estimates of the correlation function. Other estimators have been proposed which have better variance and are less sensitive to edge effects than the classic estimator of eq. 6.44. One example is the Landy–Szalay estimator (see [21]),

$$\hat{\xi}(r) = \frac{DD(r) - 2DR(r) + RR(r)}{RR(r)}. \tag{6.45}$$

The Landy–Szalay estimator can be extended to higher-order correlation functions (see [39]). For the three-point function this results in

$$\hat{\xi}(r) = \frac{DDD(r) - 3DDR(r) + DRR(r) - RRR(r)}{RRR(r)}, \tag{6.46}$$

where $DDD(r)$ represents the number of data triplets as defined by the triangular configuration shown in the central panel of figure 6.16 and $DDR(r)$, $DRR(r)$, and $RRR(r)$ are the associated configurations for the data-data-random, data-random-random, and random-random-random triplets, respectively. We note that eq. 6.46 is specified for an equilateral triangle configuration (i.e., all internal angles are held constant and the triangle configuration depends only on r). For more general triangular configurations, $DDD(r)$ and other terms depend on the lengths of all three sides of the triangle, or on the lengths of two sides and the angle between them.

AstroML implements a two-point correlation function estimator based on the Scikit-learn ball-tree:

```
import numpy as np
from astroML.correlation import two_point_angular

RA = 40 * np.random.random(1000) # RA and DEC in
     # degrees
DEC = 10 * np.random.random(1000)
bins = np.linspace(0.1, 10, 11) # evaluate in 10 bins
       # with these edges
corr = two_point_angular(RA, DEC, bins,
                         method='landy-szalay')
```

For more information, refer to the source code of figure 6.17.

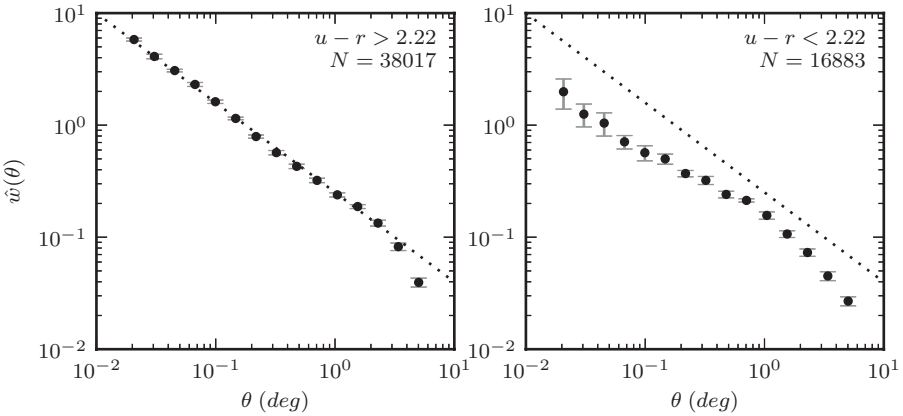


Figure 6.17. The two-point correlation function of SDSS spectroscopic galaxies in the range $0.08 < z < 0.12$, with $m < 17.7$. This is the same sample for which the luminosity function is computed in figure 4.10. Errors are estimated using ten bootstrap samples. Dotted lines are added to guide the eye and correspond to a power law proportional to $\theta^{-0.8}$. Note that the red galaxies (left panel) are clustered more strongly than the blue galaxies (right panel).

In figure 6.17, we illustrate the angular correlation function for a subset of the SDSS spectroscopic galaxy sample, in the range $0.08 < z < 0.12$. The left panel shows the correlation function for red galaxies with $u - r > 2.22$ and the right panel for blue galaxies with $u - r < 2.22$. The error bars on these plots are derived from ten bootstrap samples (i.e., independent volumes; see §4.5). Note that the clustering on small scales is much stronger for red than for blue galaxies. With 38,017 and 16,883 galaxies in the samples, a ball-tree-based implementation offers significant improvement in the computation time for the two-point correlation function over a brute-force method.

The naive computational scaling of the n-point correlation function, where we evaluate all permutations of points, is $\mathcal{O}(N^n)$ (with N the size of the data and n the order of the correlation function). For large samples of points, the computational expense of this operation can become prohibitive. Space-partitioning trees (as introduced in §2.5.2) can reduce this computational burden to $\mathcal{O}(N^{\log n})$. The underlying concept behind these efficient tree-based correlation function algorithms is the exclusion or pruning of regions of data that do not match the configuration of the correlation function (e.g., for the two-point function, pairs of points that lie outside of the range r to $r + dr$). By comparing the minimum and maximum pairwise distances between the bounding boxes of two nodes of a tree we can rapidly identify and exclude those nodes (and all child nodes) that do not match the distance constraints. This dramatically reduces the number of pairwise calculations.

A number of public implementations of tree-based correlation functions are available that utilize these techniques. These include applications optimized for single processors [29] and for parallel systems [13]. For a more thorough algorithmic discussion of computing n-point statistics, we refer the reader to WSAS.

6.6. Which Density Estimation and Clustering Algorithms Should I Use?

In this section, and the analogous sections at the end of each chapter in part III of the book, we will touch upon many of the broad issues in the "art" (as opposed to "science") of machine learning (ML), and therefore our discussions are necessarily subjective, incomplete, and cursory. However we hope this summary of unsupervised methods introduced in this chapter can serve as a springboard into the practice of ML.

We can define four axes of practical "goodness" along which one can place each machine learning method that we have described in this chapter, as well as each method in subsequent chapters:

- **Accuracy**: How well can it make accurate predictions or model data?
- **Interpretability**: How easy is it to understand why the model is making the predictions that it does, or reason about its behavior?
- **Simplicity**: Is the model "fiddly," that is, does it have numerous parameters that must be tuned and tweaked with manual effort? Is it difficult to program?
- **Speed**: Is the method fast, or is it possible via sophisticated algorithms to make it fast without altering its accuracy or other properties?

What are the most* accurate *unsupervised methods? Generally speaking, the more parameters a model has, the more flexibility it has to fit complex functions, thus reducing error. Consider first the task of *density estimation*, where the error in question is some notion of the difference between the true underlying probability density function and the estimated function. One broad division that can be made is between parametric and nonparametric methods. Parametric methods find the best model within a parametric model class, that is, one with a fixed number of model parameters. A Gaussian, or a mixture of k Gaussians, and the augmented mixture of Gaussians represented by the extreme deconvolution method are examples of a parametric model for density estimation. If the true underlying function is not a member of the chosen model class, even the model with the best setting of its parameters simply cannot fit the data. Nonparametric methods conceptually have a number of parameters that grows in some way as the number of data points N grows. Kernel density estimation is the standard nonparametric approach for density estimation in higher dimensions. Such methods can be shown to be able to fit virtually any underlying function (under very mild assumptions on the function). In general, achieving the highest accuracies requires nonparametric methods.

Note that while a mixture model (such as a mixture of Gaussians) is parametric for a fixed number of mixture components, in principle if that number is chosen in a way that can grow with N, such a model becomes nonparametric (e.g., when modeling the underlying density we can use many more components than suggested by criteria such as BIC). In practice, due to the complication and computational cost of trying every possible number of parameters, ad hoc approaches are typically used to decide the number of parameters for such models, making their status along the parametric–nonparametric scale murkier.

For the task of clustering, parametric-style options include K-means and max-radius minimization. In practice, the various heuristic procedures that are used to determine the number of clusters K often place them arguably in the

aforementioned murky regime. Mean shift, being based on kernel density estimation, is a nonparametric method in spirit, though its final step is typically similar to K-means. Hierarchical clustering is also nonparametric in spirit, in that it finds clusters at all scales, possibly $\mathcal{O}(N)$ clusters. While density estimation methods have been studied in great mathematical detail, the analogous formal statements one can make for clustering methods (e.g., regarding whether the true underlying cluster structure is recovered by a method, and if so, at what convergence rate, etc.) are much scarcer, making it difficult to speak precisely about the accuracy of clustering methods in general.

In this chapter we have also touched upon other types of unsupervised tasks such as two-sample testing. Two-sample testing using n-point correlation functions increases in accuracy as n goes up, where accuracy in this setting is related to the subtlety of the distributional differences that are detectable and the power of the test. We have not contrasted these statistics with other available statistics, but generally speaking other statistics do not extend to higher orders as the n-point statistics do.

What are the most* interpretable *methods? All of the methods we have described have the concept of a "distance" underneath them, which arguably makes them all fairly intuitive. For density estimation, mixtures of Gaussians and kernel density estimation are based on functions of Euclidean distance. For clustering, K-means, max-radius minimization, and mean shift are all based on the Euclidean distance, while hierarchical clustering uses Euclidean distance as the base notion of similarity between points and thereafter uses some distance-like notion of similarity between clusters.

The question of interpretability, in the context of many unsupervised methods, ventures into the question of validity. In clustering, as we have already touched upon, this issue is particularly acute. Most of the clustering methods, at least, can be understood as the solutions to optimization problems, that is, they find the clusterings that are optimal under some score function for clusterings. Then the clusters are valid or invalid depending on one's assessment of whether the particular score function is sensible. Hierarchical clustering is more opaque in this regard—we do not have the same sense in which the clusters it finds are the "right" ones because it is simply a procedure that begins and ends, but does not necessarily optimize any clear score function.

Among density estimation methods, the nonparametric one, kernel estimation, is perhaps the least straightforward in the sense of understanding why it works. It is in a certain sense easy to understand as a smoothing of the data by adding up "blurred" versions of the points, in the form of miniature pdfs, rather than delta functions. However, understanding why this simple method yields a legitimate nonparametric estimator of the true underlying density requires a leap into the mathematics of asymptotic statistics (see, e.g., [40]).

Parametric models, generally speaking, are more interpretable than nonparametric models because the meanings of each of the parts of the model are usually clear. A Bayesian parametric model takes this to the ultimate level by making a further level of parametric assumptions explicit. While being parametric requires assumptions which may in fact be wrong, leading to worse error, they are at least clear and understood. A nonparametric model is often effectively a collection of small

model bits, and so the effect or meaning of each part of the overall model is not clear, making nonparametric models more like black boxes.

What are the most* scalable *unsupervised methods? Max-radius minimization is generally the most efficient of the methods discussed in this chapter, requiring only K passes over the N points. K-means, for a single run (corresponding to one setting of the initial parameters and K), is quite efficient, requiring some number of iterations over the N points which is typically in the tens or hundreds. Computing Gaussian mixtures is similar to computing K-means, but each iteration is more expensive as it involves, among other things, inverting $D \times D$ covariance matrices in the evaluations of Gaussians. All of the aforementioned methods are fairly efficient because each of the N points is somehow compared to K centroids, and K is generally small, making them scale as $\mathcal{O}(N)$ per iteration with some unknown number of iterations. Kernel density estimation is similar except that the K is now replaced by N, making the computational cost quadratic in N. Mean shift, being based on KDE, is quadratic as well, and hierarchical clustering is even worse, becoming as bad as $\mathcal{O}(N^3)$ depending on the variant. Depending on the order of the statistic, n-point correlation functions are quadratic, cubic, etc. in N.

All of these methods can be sped up by fast exact (no approximation) tree-based algorithms, which generally reduce what was originally an $\mathcal{O}(N)$ cost to $\mathcal{O}(\log N)$, or an $\mathcal{O}(N^2)$ cost to $\mathcal{O}(N)$. Even $\mathcal{O}(N^3)$ costs can be reduced, in some cases to $\mathcal{O}(N \log N)$. Most of these runtimes have been proven mathematically, though some are only empirical and remain to be rigorously analyzed. The dimension of the points will affect the runtimes of such algorithms in the form of a constant exponential factor (in the dimension) in the worst case, that is, in high dimensions the algorithms will be slower. However, the quantity that matters is some notion of *intrinsic* dimension (roughly, the dimension of the submanifold upon which the data lie; see the discussion in §2.5.2), which is generally not known in advance. It is rare for a real data set not to have an intrinsic dimension which is substantially lower than its extrinsic dimension, due to the fact that dimensions tend to have some correlation with each other. In astronomical problems, even the extrinsic dimensionality is often low. When the dimensionality is too high for exact tree-based algorithms to be efficient, one can turn to approximate versions of such algorithms.

What are the* simplest *unsupervised methods to use? Max-radius minimization, not a usual staple of machine learning books, is interesting in part because of its simplicity. K-means is also one of the simplest machine learning methods to implement. Gaussian mixtures represent a step up in complexity, introducing probabilities, matrix inversions, and the resulting numerical issues. We will characterize all of these methods as somewhat "fiddly," meaning that they are not easily automatable—one must typically fiddle with the critical parameter(s), in this case K, and look at the results of many runs for each K tried, due to the fact that any particular run from an initial random starting point of the parameters may not achieve the global optimum in general. In principle, model selection for any ML method can be automated by performing cross-validation over the entire set of plausible values of the critical parameter(s) (see §6.1.1). It is more difficult in practice for these types of unsupervised problems because the best K typically does not jump out clearly from the resulting error curve.

Extreme deconvolution is an example of a Bayesian formulation of a standard ML method, in this case Gaussian mixtures. Standard ML methods, by and large, are frequentist, though for most common ones there have been Bayesianization attempts with various degrees of practical success. Broadly speaking, performing ML in a strictly Bayesian manner involves the highest amount of fiddliness for any problem, because the notion of priors adds extra distributional choices, at a minimum, generally with parameters that are explicitly not chosen in an automated data-driven manner, but by definition manually. Though simple to implement, at least at a baseline level, MCMC as a general computational workhorse adds significantly to the amount of fiddliness due to the ever-present difficulty in being sure that the Markov chain has converged, leading to many runs with different starting points and MCMC parameter settings, and frequently manual examination of the samples produced. Bayesian formulations of ML methods, extreme deconvolution being an example, which pose learning as optimization problems rather than using MCMC, can reduce this aspect of fiddliness.

Kernel density estimation, though representing an increase in statistical expressiveness over Gaussian mixtures, is simpler in that there is no parameter fitting at all, and only requires varying its critical parameter, the bandwidth h. Mean shift requires KDE-like steps in its optimization iterations, followed typically by K-means, putting it in the fiddly category. Hierarchical clustering is arguably both more powerful and simpler than methods like K-means that require selection of K because it finds clustering at all scales, with no parameter fitting or random initializations, finding the same clusters each time it is run.

Other considerations, and taste Other considerations include problem-specific circumstances which some methods may be able to handle more naturally or effectively than others. For example, probabilistic models that can be optimized via the EM algorithm, such as Gaussian mixtures, can be naturally augmented to perform missing value imputation, that is, using the best model so far to fill in guesses for any missing entries in the data matrix. Extreme deconvolution is a specific augmentation of Gaussian mixtures to incorporate measurement errors, while standard formulations of ML methods do not come with a way to incorporate errors.

Often in astronomy, physical insight may also play a role in the choice of analysis method. For example, the two-point correlation function is a mathematical transformation of the power spectrum, the expected form of which can be computed (at least in the simplest linear regime) analytically from first principles.

In reality the final consideration, though often not admitted or emphasized as much as practice reflects, is *taste*. A strong branching in taste often occurs at the decision of whether to join either the Bayesian or frequentist camp. The more common decision is to treat the set of available paradigms and methods as a smorgasbord from which to choose, according to the trade-offs they represent and the characteristics of one's problem. But in practice several different methods may not be strongly distinguishable, at least not initially, from a pure performance point of view. At that point, one may simply be attracted to the "style" or connotations of a method, such as the MLE roots of Gaussian mixtures, or the nonparametric theory roots of KDE.

TABLE 6.1.
Summary of the practical properties of different unsupervised techniques.

Method	Accuracy	Interpretability	Simplicity	Speed
K-nearest-neighbor	H	H	H	M
Kernel density estimation	H	H	H	H
Gaussian mixture models	H	M	M	M
Extreme deconvolution	H	H	M	M
K-means	L	M	H	M
Max-radius minimization	L	M	M	M
Mean shift	M	H	H	M
Hierarchical clustering	H	L	L	L
Correlation functions	H	M	M	M

Simple summary Table 6.1 is a quick summary of our assessment of each of the methods considered in this chapter, in terms of high (H), medium (M), and low (L). Note that the table mixes methods for different tasks (density estimation, clustering, correlation functions), so they are not all directly comparable to each other.

References

[1] Allison, R. J., S. P. Goodwin, R. J. Parker, and others (2009). Using the minimum spanning tree to trace mass segregation. *MNRAS 395*, 1449–1454.

[2] Audit, E., R. Teyssier, and J.-M. Alimi (1998). Non-linear dynamics and mass function of cosmic structures. II. Numerical results. *A&A 333*, 779–789.

[3] Balogh, M., V. Eke, C. Miller, and others (2004). Galaxy ecology: Groups and low-density environments in the SDSS and 2dFGRS. *MNRAS 348*, 1355–1372.

[4] Barrow, J. D., S. P. Bhavsar, and D. H. Sonoda (1985). Minimal spanning trees, filaments and galaxy clustering. *MNRAS 216*, 17–35.

[5] Bovy, J., J. Hennawi, D. Hogg, and others (2011). Think outside the color box: Probabilistic target selection and the SDSS-XDQSO Quasar Targeting Catalog. *ApJ 729*, 141.

[6] Bovy, J., D. Hogg, and S. Roweis (2011). Extreme deconvolution: Inferring complete distribution functions from noisy, heterogeneous and incomplete observations. *The Annals of Applied Statistics 5*(2B), 1657–1677.

[7] Casertano, S. and P. Hut (1985). Core radius and density measurements in N-body experiments. Connections with theoretical and observational definitions. *ApJ 298*, 80–94.

[8] Connolly, A. J., R. Scranton, D. Johnston, and others (2002). The angular correlation function of galaxies from early Sloan Digital Sky Survey data. *ApJ 579*, 42–47.

[9] Davis, M., G. Efstathiou, C. S. Frenk, and S. White (1985). The evolution of large-scale structure in a universe dominated by cold dark matter. *ApJ 292*, 371–394.

[10] Delaigle, A. and A. Meister (2008). Density estimation with heteroscedastic error. *Bernoulli 14*(2), 562–579.

[11] Dressler, A. (1980). Galaxy morphology in rich clusters—implications for the formation and evolution of galaxies. *ApJ 236*, 351–365.

[12] Ferdosi, B. J., H. Buddelmeijer, S. C. Trager, M. Wilkinson, and J. Roerdink (2011). Comparison of density estimation methods for astronomical datasets. *A&A 531*, A114.

[13] Gardner, J. P., A. Connolly, and C. McBride (2007). Enabling rapid development of parallel tree search applications. In *Proceedings of the 5th IEEE Workshop on Challenges of Large Applications in Distributed Environments*, CLADE '07, New York, NY, USA, pp. 1–10. ACM.

[14] Gott, III, J. R., M. Jurić, D. Schlegel, and others (2005). A map of the universe. *ApJ 624*, 463–484.

[15] Gray, A. G. and A. W. Moore (2003). Nonparametric density estimation: Toward computational tractability. In *SIAM International Conference on Data Mining (SDM)*.

[16] Ivezić, Ž., R. H. Lupton, D. Schlegel, and others (2004). SDSS data management and photometric quality assessment. *Astronomische Nachrichten 325*, 583–589.

[17] Ivezić, Ž., J. A. Smith, G. Miknaitis, and others (2007). Sloan Digital Sky Survey Standard Star Catalog for Stripe 82: The dawn of industrial 1% optical photometry. *AJ 134*, 973–998.

[18] Ivezić, Ž., A. K. Vivas, R. H. Lupton, and R. Zinn (2005). The selection of RR Lyrae stars using single-epoch data. *AJ 129*, 1096–1108.

[19] Kruskal, J. B. (1956). On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society 7*(1), 48–50.

[20] Krzewina, L. G. and W. C. Saslaw (1996). Minimal spanning tree statistics for the analysis of large-scale structure. *MNRAS 278*, 869–876.

[21] Landy, S. D. and A. S. Szalay (1993). Bias and variance of angular correlation functions. *ApJ 412*, 64–71.

[22] Lee, D. and A. G. Gray (2006). Faster Gaussian summation: Theory and experiment. In *Conference on Uncertainty in Artificial Intelligence (UAI)*.

[23] Lee, D. and A. G. Gray (2009). Fast high-dimensional kernel summations using the Monte Carlo multipole method. In *Advances in Neural Information Processing Systems (NIPS) (Dec 2008)*. MIT Press.

[24] Lee, D., A. G. Gray, and A. W. Moore (2006). Dual-tree fast gauss transforms. In Y. Weiss, B. Schölkopf, and J. Platt (Eds.), *Advances in Neural Information Processing Systems (NIPS) (Dec 2005)*. MIT Press.

[25] Lee, D., R. Vuduc, and A. G. Gray (2012). A distributed kernel summation framework for general-dimension machine learning. In *SIAM International Conference on Data Mining (SDM)*.

[26] Limber, D. N. (1953). The analysis of counts of the extragalactic nebulae in terms of a fluctuating density field. *ApJ 117*, 134.

[27] March, W. B., P. Ram, and A. G. Gray (2010). Fast Euclidean minimum spanning tree: algorithm, analysis, and applications. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '10, pp. 603–612. ACM.

[28] Martínez, V. and E. Saar (2002). *Statistics of the Galaxy Distribution*. Chapman and Hall/CRC.

[29] Moore, A., A. Connolly, C. Genovese, and others (2000). Fast algorithms and efficient statistics: N-point correlation functions. *Mining the Sky 370*(2), 71–+.

[30] Peebles, P. (1980). *The Large-Scale Structure of the Universe*. Princeton Series in Physics. Princeton University Press.

[31] Press, W. H. and M. Davis (1982). How to identify and weigh virialized clusters of galaxies in a complete redshift catalog. *ApJ 259*, 449–473.

[32] Prim, R. C. (1957). Shortest connection networks and some generalizations. *Bell System Technology Journal 36*, 1389–1401.

[33] Ram, P., D. Lee, W. March, and A. G. Gray (2010). Linear-time algorithms for pairwise statistical problems. In *Advances in Neural Information Processing Systems (NIPS) (Dec 2009)*. MIT Press.

[34] Roche, A. (2011). EM algorithm and variants: An informal tutorial. *ArXiv:statistics/1105.1476*.

[35] Scott, D. W. (2008). *Multivariate Density Estimation: Theory, Practice, and Visualization*. John Wiley & Sons, Inc., Hoboken, NJ, USA.

[36] Sesar, B., Ž. Ivezić, S. H. Grammer, and others (2010). Light curve templates and galactic distribution of RR Lyrae stars from Sloan Digital Sky Survey Stripe 82. *ApJ 708*, 717–741.

[37] Silverman, B. (1986). *Density Estimation for Statistics and Data Analysis*. Monographs on Statistics and Applied Probability. Chapman and Hall.

[38] Stefanski, L. and J. Raymond (1990). Deconvolving kernel density estimators. *Statistics: A Journal of Theoretical and Applied Statistics 21*(2), 169–184.

[39] Szapudi, S. and A. S. Szalay (1998). A new class of estimators for the N-point correlations. *ApJL 494*, L41.

[40] van der Vaart, A. W. (2000). *Asymptotic Statistics*. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press.

[41] Wand, M. P. and M. C. Jones (1995). Kernel smoothing. Chapman and Hall.

7 Dimensionality and Its Reduction

"A mind that is stretched by a new idea can never go back to its original dimensions." (Oliver Wendell Holmes)

With the dramatic increase in data available from a new generation of astronomical telescopes and instruments, many analyses must address the question of the complexity as well as size of the data set. For example, with the SDSS imaging data we could measure arbitrary numbers of properties or features for any source detected on an image (e.g., we could measure a series of progressively higher moments of the distribution of fluxes in the pixels that make up the source). From the perspective of efficiency we would clearly rather measure only those properties that are directly correlated with the science we want to achieve. In reality we do not know the correct measurement to use or even the optimal set of functions or bases from which to construct these measurements. This chapter deals with how we can learn which measurements, properties, or combinations thereof carry the most information within a data set. The techniques we will describe here are related to concepts we have discussed when describing Gaussian distributions (§3.5.2), density estimation (§6.1), and the concepts of information content (§5.2.2).

We will start in §7.1 with an exploration of the problems posed by high-dimensional data. In §7.2 we will describe the data sets used in this chapter, and in §7.3, we will introduce perhaps the most important and widely used dimensionality reduction technique, principal component analysis (PCA). In the remainder of the chapter, we will introduce several alternative techniques which address some of the weaknesses of PCA.

7.1. The Curse of Dimensionality

Imagine that you have decided to purchase a car and that your initial thought is to purchase a "fast" car. Whatever the exact implementation of your search strategy, let us assume that your selection results in a fraction $r < 1$ of potential matches. If you expand the requirements for your perfect car such that it is "red," has "8 cylinders," and a "leather interior" (with similar selection probabilities), then your selection probability would be r^4. If you also throw "classic car" into the mix, the selection probability becomes r^5. The more selection conditions you adopt, the tinier is the

chance of finding your ideal car! These selection conditions are akin to dimensions in your data set, and this effect is at the core of the phenomenon known as the "curse of dimensionality"; see [2].

From a data analysis point of view, the curse of dimensionality impacts the size of data required to constrain a model, the complexity of the model itself, and the search times required to optimize the model. Considering just the first of these issues, we can quantitatively address the question of how much data is needed to estimate the probability of finding points within a high-dimensional space. Imagine that you have drawn N points from a D-dimensional uniform distribution, described by a hypercube (centered at the origin) with edge length 2 (i.e., each coordinate lies within the range $[-1, 1]$). Under a Euclidean distance metric, what proportion of points fall within a unit distance of the origin?

If the points truly are uniformly distributed throughout the volume, then a good estimate of this is the ratio of the volume of a unit hypersphere centered at the origin, to the volume of the side-length=2 hypercube centered at the origin. For two dimensions, this gives

$$f_2 = \frac{\pi r^2}{(2r)^2} = \pi/4 \approx 78.5\%. \tag{7.1}$$

For three dimensions, this becomes

$$f_3 = \frac{(4/3)\pi r^3}{(2r)^3} = \pi/6 \approx 52.3\%. \tag{7.2}$$

Generalizing to higher dimensions requires some less familiar formulas for the hypervolumes of D-spheres. It can be shown that the hypervolume of a D-dimensional hypersphere with radius r is given by

$$V_D(r) = \frac{2r^D \pi^{D/2}}{D\,\Gamma(D/2)}, \tag{7.3}$$

where $\Gamma(z)$ is the complete gamma function. The reader can check that evaluating this for $D = 2$ and $D = 3$ yield the familiar formulas for the area of a circle and the volume of a sphere. With this formula we can compute

$$f_D = \frac{V_D(r)}{(2r)^D} = \frac{\pi^{D/2}}{D2^{D-1}\Gamma(D/2)}. \tag{7.4}$$

One can quite easily show that

$$\lim_{D\to\infty} f_D = 0. \tag{7.5}$$

That is, in the limit of many dimensions, *not a single point* will be within a unit radius of the origin! In other words, the fraction of points within a search radius r (relative to the full space) will tend to zero as the dimensionality grows and, therefore, the number of points in a data set required to evenly sample this hypervolume will grow exponentially with dimension. In the context of astronomy, the SDSS [1] comprises

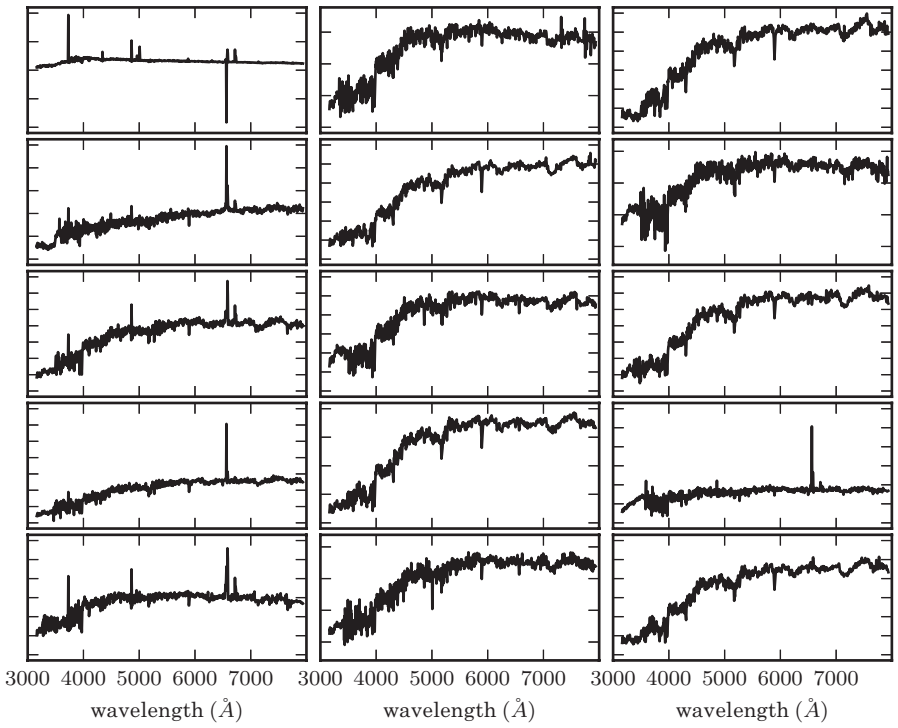


Figure 7.1. A sample of 15 galaxy spectra selected from the SDSS spectroscopic data set (see §1.5.5). These spectra span a range of galaxy types, from star-forming to passive galaxies. Each spectrum has been shifted to its rest frame and covers the wavelength interval 3000–8000 Å. The specific fluxes, $F_\lambda(\lambda)$, on the ordinate axes have an arbitrary scaling.

a sample of 357 million sources. Each source has 448 measured attributes (e.g., measures of flux, size, shape, and position). If we used our physical intuition to select just 30 of those attributes from the database (e.g., a subset of the magnitude, size, and ellipticity measures) and normalized the data such that each dimension spanned the range -1 to 1, the probability of having one of the 357 million sources reside within the unit hypersphere would be only one part in 1.4×10^5.

Given the dimensionality of current astronomical data sets, how can we ever hope to find or characterize any structure that might be present? The underlying assumption behind our earlier discussions has been that all dimensions or attributes are created equal. We know that is, however, not true. There exist projections within the data that capture the principal physical and statistical correlations between measured quantities (this idea lies behind the *intrinsic dimensionality* discussed in §2.5.2). Finding these dimensions or axes efficiently and thereby reducing the dimensionality of the underlying data is the subject of this chapter.

7.2. The Data Sets Used in This Chapter

Throughout this chapter we use the SDSS galaxy spectra described in §1.5.4 and §1.5.5, as a proxy for high-dimensional data. Figure 7.1 shows a representative sample

of these spectra covering the interval 3200–7800 Å in 1000 wavelength bins. While a spectrum defined as $x(\lambda)$ may not immediately be seen as a point in a high-dimensional space, it can be represented as such. The function $x(\lambda)$ is in practice sampled at D discrete flux values, and written as a D-dimensional vector. And just as a three-dimensional vector is often visualized as a point in a three-dimensional space, this spectrum (represented by a D-dimensional vector) can be thought of as a single point in D-dimensional space. Analogously, a $D = N \times K$ image may also be expressed as a vector with D elements, and therefore a point in a D-dimensional space. So, while we use spectra as our proxy for high-dimensional space, the algorithms and techniques described in this chapter are applicable data as diverse as catalogs of multivariate data, two-dimensional images, and spectral hypercubes.

7.3. Principal Component Analysis

Figure 7.2 shows a two-dimensional distribution of points drawn from a Gaussian centered on the origin of the x- and y-axes. While the points are strongly correlated along a particular direction, it is clear that this correlation does not align with the initial choice of axes. If we wish to reduce the number of features (i.e., the number of axes) that are used to describe these data (providing a more compact representation) then it is clear that we should rotate our axes to align with this correlation (we have already encountered this rotation in eq. 3.82). Any rotation preserves the relative ordering or configuration of the data so we choose our rotation to maximize the ability to discriminate between the data points. This is accomplished if the rotation maximizes the variance along the resulting axes (i.e., defining the first axis, or principal component, to be the direction with maximal variance, the second principal component to be orthogonal to the first component that maximizes the residual variance, and so on). As indicated in figure 7.2, this is mathematically equivalent to a regression that minimizes the square of the orthogonal distances from the points to the principal axes.

This dimensionality reduction technique is known as a principal component analysis (PCA). It is also referred to in the literature as a Karhunen–Loéve [21, 25] or Hotelling transform. PCA is a linear transform, applied to multivariate data, that defines a set of uncorrelated axes (the principal components) ordered by the variance captured by each new axis. It is one of the most widely applied dimensionality reduction techniques used in astrophysics today and dates back to Pearson who, in 1901, developed a procedure for fitting lines and planes to multivariate data; see [28].

There exist a number of excellent texts on PCA that review its use across a broad range of fields and applications (e.g., [19] and references therein). We will, therefore, focus our discussion of PCA on a brief description of its mathematical formalism then concentrate on its application to astronomical data and its use as a tool for classification, data compression, regression, and signal-to-noise filtering of high-dimensional data sets.

Before progressing further with the application of PCA, it is worth noting that many of the applications of PCA to astronomical data describe the importance of the orthogonal nature of PCA (i.e., the ability to project a data set onto a set of uncorrelated axes). It is often forgotten that the observations themselves are already a

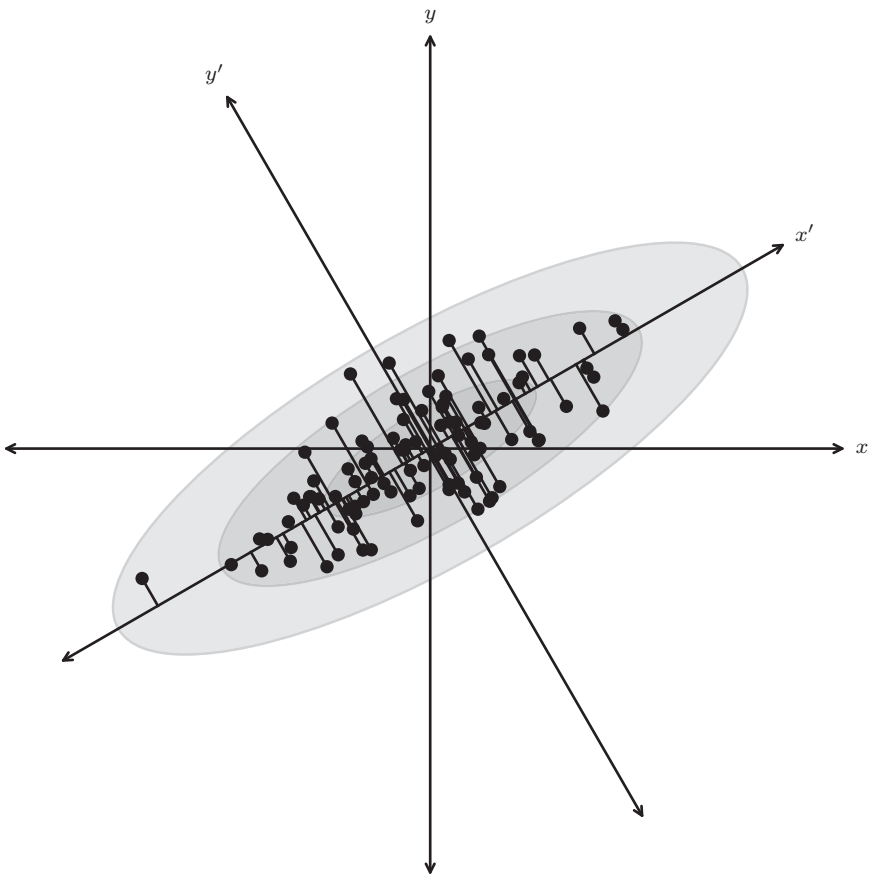


Figure 7.2. A distribution of points drawn from a bivariate Gaussian and centered on the origin of x and y. PCA defines a rotation such that the new axes (x' and y') are aligned along the directions of maximal variance (the principal components) with zero covariance. This is equivalent to minimizing the square of the perpendicular distances between the points and the principal components.

representation of an orthogonal basis (e.g., the axes {1,0,0,0, ...}, {0,1,0,0,0, ...}, etc.). As we will show, the importance of PCA is that *the new axes are aligned with the direction of maximum variance within the data* (i.e., the direction with the maximum signal).

7.3.1. The Derivation of Principal Component Analyses

Consider a set of data, $\{x_i\}$, comprising a series of N observations with each observation made up of K measured features (e.g., size, color, and luminosity, or the wavelength bins in a spectrum). We initially center the data by subtracting the mean of each feature in $\{x_i\}$ and then write this $N \times K$ matrix as X.[1] The covariance

[1]Often the opposite convention is used: that is, N points in K dimensions are stored in a $K \times N$ matrix rather than an $N \times K$ matrix. We choose the latter to align with the convention used in Scikit-learn and AstroML.

of the centered data, C_X, is given by

$$C_X = \frac{1}{N-1} X^T X, \tag{7.6}$$

where the $N - 1$ term comes from the fact that we are working with the sample covariance matrix (i.e., the covariances are derived from the data themselves).

Nonzero off-diagonal components within the covariance matrix arise because there exist correlations between the measured features (as we saw in figure 7.2; recall also the discussion of bivariate and multivariate distributions in §3.5). PCA wishes to identify a projection of $\{x_i\}$, say, R, that is aligned with the directions of maximal variance. We write this projection as $Y = XR$ and its covariance as

$$C_Y = R^T X^T X R = R^T C_X R \tag{7.7}$$

with C_X the covariance of X as defined above.

The first principal component, r_1, of R is defined as the projection with the maximal variance (subject to the constraint that $r_1^T r_1 = 1$). We can derive this principal component by using Lagrange multipliers and defining the cost function, $\phi(r_1, \lambda)$, as

$$\phi(r_1, \lambda_1) = r_1^T C_X r_1 - \lambda_1 (r_1^T r_1 - 1). \tag{7.8}$$

Setting the derivative of $\phi(r_1, \lambda)$ (with respect to r_1) to zero gives

$$C_X r_1 - \lambda_1 r_1 = 0. \tag{7.9}$$

λ_1 is, therefore, the root of the equation $\det(C_X - \lambda_1 \mathbf{I}) = 0$ and is an eigenvalue of the covariance matrix. The variance for the first principal component is maximized when

$$\lambda_1 = r_1^T C_X r_1 \tag{7.10}$$

is the largest eigenvalue of the covariance matrix. The second (and further) principal components can be derived in an analogous manner by applying the additional constraint to the cost function that the principal components are uncorrelated (e.g., $r_2^T C_X r_1 = 0$).

The columns of R are then the eigenvectors or principal components, and the diagonal values of C_Y define the amount of variance contained within each component. With

$$C_X = R C_Y R^T \tag{7.11}$$

and ordering the eigenvectors by their eigenvalue we can define the set principal components for X.

Efficient computation of principal components

One of the most direct methods for computing the PCA is through the eigenvalue decomposition of the covariance or correlation matrix, or equivalently through the

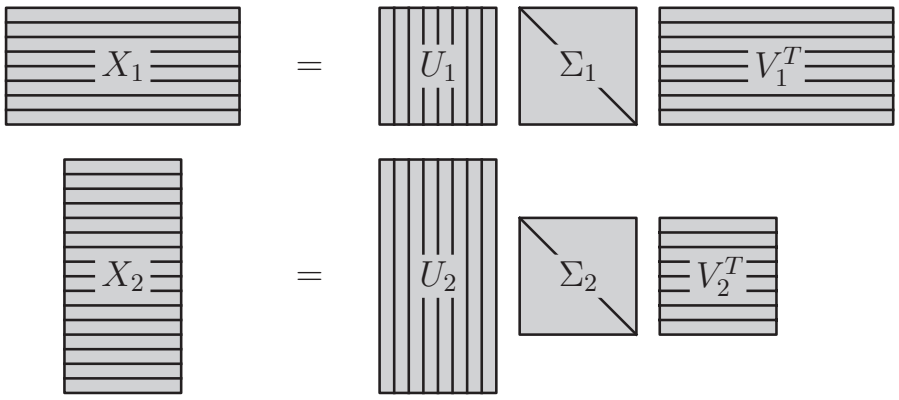


Figure 7.3. Singular value decomposition (SVD) can factorize an $N \times K$ matrix into $U\Sigma V^T$. There are different conventions for computing the SVD in the literature, and this figure illustrates the convention used in this text. The matrix of singular values Σ is always a square matrix of size $[R \times R]$ where $R = \min(N, K)$. The shape of the resulting U and V matrices depends on whether N or K is larger. The columns of the matrix U are called the left-singular vectors, and the columns of the matrix V are called the right-singular vectors. The columns are orthonormal bases, and satisfy $U^TU = V^TV = I$.

singular value decomposition (SVD) of the data matrix itself. The scaled SVD can be written

$$U\Sigma V^T = \frac{1}{\sqrt{N-1}}X, \tag{7.12}$$

where the columns of U are the *left-singular vectors*, and the columns of V are the *right-singular vectors*. There are many different conventions for the SVD in the literature; we will assume the convention that the matrix of singular values Σ is always a square, diagonal matrix, of shape $[R \times R]$ where $R = \min(N, K)$ is the rank of the matrix X (assuming all rows and columns of X are independent). U is then an $[N \times R]$ matrix, and V^T is an $[R \times K]$ matrix (see figure 7.3 for a visualization of this SVD convention). The columns of U and V form orthonormal bases, such that $U^TU = V^TV = I$.

Using the expression for the covariance matrix (eq. 7.6) along with the scaled SVD (eq. 7.12) gives

$$\begin{aligned} C_X &= \left[\frac{1}{\sqrt{N-1}}X\right]^T \left[\frac{1}{\sqrt{N-1}}X\right] \\ &= V\Sigma U^T U\Sigma V^T \\ &= V\Sigma^2 V^T. \end{aligned} \tag{7.13}$$

Comparing to eq 7.11, we see that the right singular vectors V correspond to the principal components R, and the diagonal matrix of eigenvalues C_Y is equivalent to the square of the singular values,

$$\Sigma^2 = C_Y. \tag{7.14}$$

Thus the eigenvalue decomposition of C_X, and therefore the principal components, can be computed from the SVD of X, without explicitly constructing the matrix C_X.

NumPy and SciPy contain powerful suites of linear algebra tools. For example, we can confirm the above relationship using `svd` for computing the SVD, and `eigh` for computing the symmetric (or in general Hermitian) eigenvalue decomposition:

```
>>> import numpy as np
>>> X = np.random.random((100, 3))
>>> CX = np.dot(X.T, X)
>>> U, Sdiag, VT = np.linalg.svd(X, full_matrices=
    False)
>>> CYdiag, R = np.linalg.eigh(CX)
```

The `full_matrices` keyword assures that the convention shown in figure 7.3 is used, and for both Σ and C_Y, only the diagonal elements are returned. We can compare the results, being careful of the different ordering conventions: `svd` puts the largest singular values first, while `eigh` puts the smallest eigenvalues first:

```
>>> np.allclose(CYdiag, Sdiag[::-1] ** 2)
 # [::-1] reverses the array
True
>>> np.set_printoptions(suppress=True)
 # clean output for below
>>> VT[::-1].T / R
array([[-1., -1.,  1.],
       [-1., -1.,  1.],
       [-1., -1.,  1.]])
```

The eigenvectors of C_X and the right singular vectors of X agree up to a sign, as expected. For more information, see appendix A or the documentation of `numpy.linalg` and `scipy.linalg`.

The SVD formalism can also be used to quickly see the relationship between the covariance matrix C_X, and the correlation matrix,

$$\begin{aligned} M_X &= \frac{1}{N-1} X X^T \\ &= U \Sigma V^T V \Sigma U^T \\ &= U \Sigma^2 U^T \end{aligned} \tag{7.15}$$

in analogy with above. The left singular vectors, U, turn out to be the eigenvectors of the correlation matrix, which has eigenvalues identical to those of the covariance matrix. Furthermore, the orthonormality of the matrices U and V means that if U is known, V (and therefore R) can be quickly determined using the linear algebraic

manipulation of eq. 7.12:

$$R = V = \frac{1}{\sqrt{N-1}} X^T U \Sigma^{-1}. \tag{7.16}$$

Thus we have three equivalent ways of computing the principal components R and the eigenvalues C_X: the SVD of X, the eigenvalue decomposition of C_X, or the eigenvalue decomposition of M_X. The optimal procedure will depend on the relationship between the data size N and the dimensionality K. If $N \gg K$, then using the eigenvalue decomposition of the $K \times K$ covariance matrix C_X will in general be more efficient. If $K \gg N$, then using the $N \times N$ correlation matrix M_X will be more efficient. In the intermediate case, direct computation of the SVD of X will be the most efficient route.

7.3.2. The Application of PCA

PCA can be performed easily using Scikit-learn:

```
import numpy as np
from sklearn.decomposition import PCA

X = np.random.normal(size=(100, 3))
    # 100 points in 3 dimensions
R = np.random.random((3, 10))  # projection matrix
X = np.dot(X, R)  # X is now 10-dim, with 5 intrinsic
    # dims
pca = PCA(n_components=4)  # n_components can be
      # optionally set
pca.fit(X)
comp = pca.transform(X)  # compute the subspace
      # projection of X

mean = pca.mean_  # length 10 mean of the data
components = pca.components_  # 4 x 10 matrix of
             # components
var = pca.explained_variance_  # the length 4 array
      # of eigenvalues
```

In this case, the last element of `var` will be zero, because the data is inherently three-dimensional. For larger problems, `RandomizedPCA` is also useful. For more information, see the Scikit-learn documentation.

To form the data matrix X, the data vectors are centered by subtracting the mean of each dimension. Before this takes place, however, the data are often preprocessed to ensure that the PCA is maximally informative. In the case of heterogeneous data (e.g., galaxy shape and flux), the columns are often preprocessed by dividing by

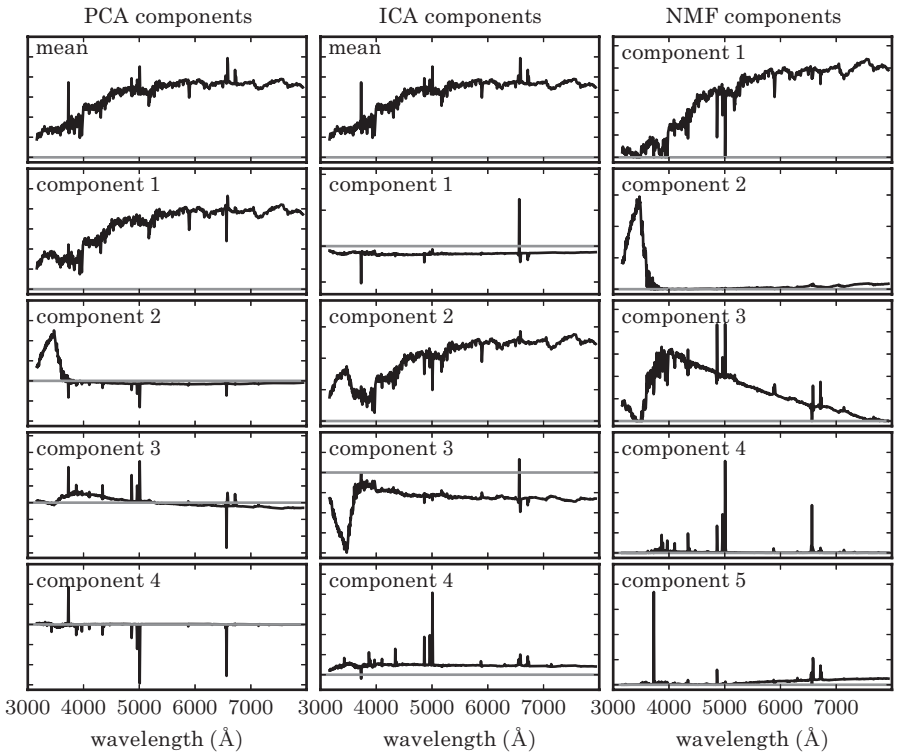


Figure 7.4. A comparison of the decomposition of SDSS spectra using PCA (left panel—see §7.3.1), ICA (middle panel—see §7.6) and NMF (right panel—see §7.4). The rank of the component increases from top to bottom. For the ICA and PCA the first component is the mean spectrum (NMF does not require mean subtraction). All of these techniques isolate a common set of spectral features (identifying features associated with the continuum and line emission). The ordering of the spectral components is technique dependent.

their variance. This so-called whitening of the data ensures that the variance of each feature is comparable, and can lead to a more physically meaningful set of principal components. In the case of spectra or images, a common preprocessing step is to normalize each row, such that the integrated flux of each object is one. This helps to remove uninteresting correlations based on the overall brightness of the spectrum or image.

For the case of the galaxy spectra in figure 7.1, each spectrum has been normalized to a constant total flux, before being centered such that the spectrum has zero mean (this subtracted mean spectrum is shown in the upper-left panel of figure 7.4). The principal directions found in the high-dimensional data set are often referred to as the "eigenspectra," and just as a vector can be represented by the sum of its components, a spectrum can be represented by the sum of its eigenspectra. The left panel of figure 7.4 shows, from top to bottom, the mean spectrum and the first four eigenspectra. The eigenspectra are ordered by their associated eigenvalues shown in figure 7.5. Figure 7.5 is often referred to as a scree plot (related to the shape of rock debris after it has fallen down a slope; see [6]) with the eigenvalues reflecting

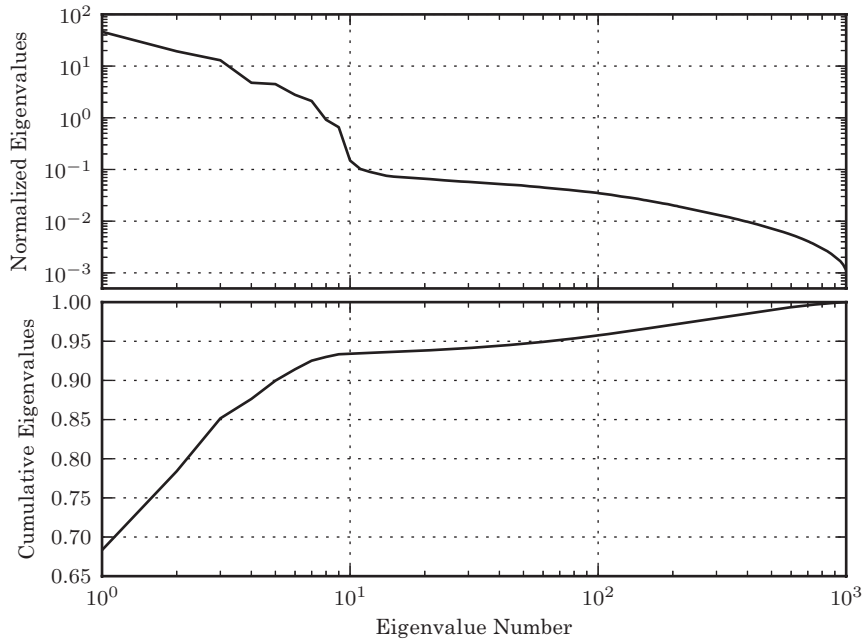


Figure 7.5. The eigenvalues for the PCA decomposition of the SDSS spectra described in §7.3.2. The top panel shows the decrease in eigenvalue as a function of the number of eigenvectors, with a break in the distribution at ten eigenvectors. The lower panel shows the cumulative sum of eigenvalues normalized to unity. 94% of the variance in the SDSS spectra can be captured using the first ten eigenvectors.

the amount of variance contained within each of the associated eigenspectra (with the constraint that the sum of the eigenvalues equals the total variance of the system).

The cumulative variance associated with the eigenvectors measures the amount of variance of the *entire data set* which is encoded in the eigenvectors. From figure 7.5, we see that ten eigenvectors are responsible for 94% of the variance in the sample: this means that by projecting each spectrum onto these first ten eigenspectra, an average of 94% of the "information" in each spectrum is retained, where here we use the term "information" loosely as a proxy for variance. This amounts to a compression of the data by a factor of 100 (using ten of the 1000 eigencomponents) with a very small loss of information. This is the sense in which PCA allows for dimensionality reduction.

This concept of data compression is supported by the shape of the eigenvectors. Eigenvectors with large eigenvalues are predominantly low-order components (in the context of astronomical data they primarily reflect the continuum shape of the galaxies). Higher-order components (with smaller eigenvalues) are predominantly made up of sharp features such as emission lines. The combination of continuum and line emission within these eigenvectors can describe any of the input spectra. The remaining eigenvectors reflect the noise within the ensemble of spectra in the sample.

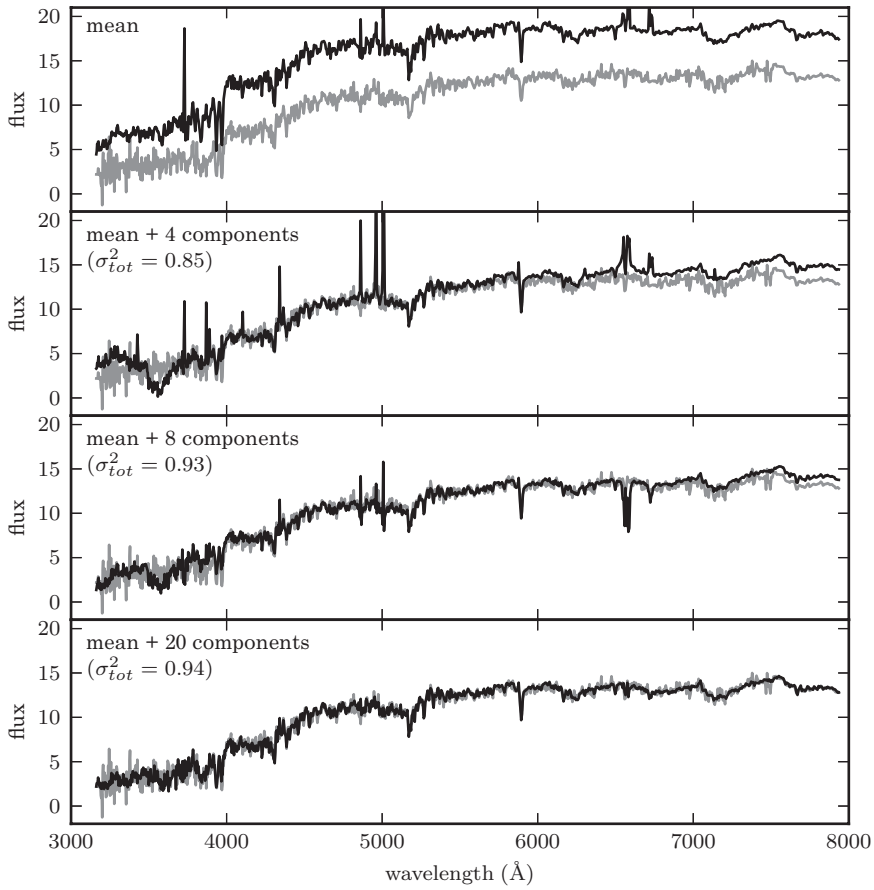


Figure 7.6. The reconstruction of a particular spectrum from its eigenvectors. The input spectrum is shown in gray, and the partial reconstruction for progressively more terms is shown in black. The top panel shows only the mean of the set of spectra. By the time 20 PCA components are added, the reconstruction is very close to the input, as indicated by the expected total variance of 94%.

The reconstruction of an example spectrum, $\boldsymbol{x}(k)$, from the eigenbasis, $\boldsymbol{e}_i(k)$ is shown in figure 7.6. Each spectrum $\boldsymbol{x}_i(k)$ can be described by

$$\boldsymbol{x}_i(k) = \boldsymbol{\mu}(k) + \sum_j^R \theta_{ij} \boldsymbol{e}_j(k), \tag{7.17}$$

where i represents the number of the input spectrum, j represents the number of the eigenspectrum, and, for the case of a spectrum, k represents the wavelength. Here, $\boldsymbol{\mu}(k)$ is the mean spectrum and θ_{ij} are the linear expansion coefficients derived from

$$\theta_{ij} = \sum_k \boldsymbol{e}_j(k)(\boldsymbol{x}_i(k) - \boldsymbol{\mu}(k)). \tag{7.18}$$

R is the total number of eigenvectors (given by the rank of X, $\min(N,K)$). If the summation is over all eigenvectors, the input spectrum is fully described with no loss of information. Truncating this expansion (i.e., $r < R$),

$$\boldsymbol{x}_i(k) = \sum_{i}^{r<R} \theta_i \boldsymbol{e}_i(k), \tag{7.19}$$

will exclude those eigencomponents with smaller eigenvalues. These components will, predominantly, reflect the noise within the data set. This is reflected in figure 7.6: truncating the reconstruction at 20 components captures the overall shape and important features of the spectrum: the differences between the reconstruction and the input spectrum are mostly high-frequency spectral noise.

A number of aspects of PCA are worth noting. Comparisons between the eigenvectors derived from PCA and known spectral types of galaxies have shown that these statistically orthogonal components correlate strongly with specific physical properties (i.e., they relate to the star formation and the composition of the stellar types within a galaxy spectrum, e.g., [33, 34]). Second, given the cumulative nature of the sum of variances used in PCA, astrophysically interesting components within the spectra (e.g., sharp spectral lines or transient features for certain galaxy populations) may not be reflected in the largest PCA components. Because of this, care must be taken when truncating at a small number of components. Additionally, the assumption that a sum of linear components can efficiently reconstruct the features within the data does not always hold. An example of this is the variation in broad emission lines (such as those from quasars). The variation in line width is an inherently nonlinear process and can require a large number of components to fully characterize: for broad line quasars over 30 components are required to reproduce the underlying spectra compared to the 10 required for quiescent and star-forming galaxies. In these cases, dimensionality reduction techniques based on the local structure need to be considered (see §7.5). Finally, up to this point we have ignored errors and missing data when considering the application of PCA. We address this in §7.3.3.

Choosing the Level of Truncation in an Expansion

One of the critical issues when reconstructing a data set from a linear combination of eigenvectors is choosing the number of components, r, to keep. Too many components will introduce noise into the reconstruction. Too few may not capture the complete physical correlations within the data. While many attempts have been made to place the choice of r on a sound statistical footing, the techniques that are used today are typically either based on empirical relations derived from simplified experiments or derived from a series of somewhat ad hoc assumptions (see [19] for a detailed discussion).

The most common criterion for defining r is based on the total variance captured in the first r eigenvectors. If we specify a bound, α, on the fraction of the variance we wish to capture then we can define r from the summation of the

eigenvalues, σ_i, that are the diagonals of the matrix Σ:

$$\frac{\sum_i^{i=r} \sigma_i}{\sum_i^{i=R} \sigma_i} < \alpha. \tag{7.20}$$

Typical values for α range from 0.70 to 0.95, though the choice of threshold is sensitive to the shape of the scree plot (figure 7.5); for a shallow slope in the scree plot, whether r or $r + 1$ crosses the threshold is somewhat arbitrary.

The shape of the scree plot can be used to define the level of truncation. Cattell [6], using factor analysis, proposed that a sharp change in the gradient of the eigenvalues (i.e., a knee in the scree plot) could be used to define the cutoff value, r. The knee is defined by [6] as $\Sigma_r^2 - \Sigma_{r+1}^2$. If no clearly defined break in the scree plot exists, the definition of this cutoff becomes problematic. A modification of the technique in [6] is the LEV diagram, which plots the logarithm of the eigenvalue against the number of eigenvectors. The rationale for this modification is that if noise decays geometrically, variation in the eigenvalues should drop as a linear function.

For the correlation matrix PCA, Kaiser's rule [20] or the Guttman–Kaiser criterion can be applied. This states that, if all of the elements of x are independent then all principal components would have unit variance. In this case, r can be set to the limit where the eigenvalues in the scree plot fall below unity. In the context of the covariance matrix this can be reformulated as the setting of r to the number of components at which the eigenvalue falls to the average of all eigenvalues. Jolliffe [19] proposed a modification of this truncation to 70% of the average eigenvalues to allow for sample variance (which increases the number of components returned). Experiments with Kaiser's rule show that it tends to overpredict the number of components that remain after truncation.

Each of the criteria described above are sensitive to the shape of the scree plot and the choice of truncation rule remains application specific.

7.3.3. PCA with Missing Data

Until now we have assumed that the data we are working with are complete, without gaps or censored elements. In real-world applications, the presence of detector glitches, variable noise, or masking effects (e.g., sky lines in astronomical spectra) can make these assumptions invalid. Truncation of the expansion provides a signal-to-noise filtering of the data. The PCA bases should also be able to correct for missing elements within the data: because the PCA components encode the correlation of each flux with the other measured fluxes, these components should provide a natural way to determine these missing values.

One complication we must address is that eigenspectra are only defined to be orthogonal over the data range on which they are constructed. If a data vector does not fully cover that space then projecting the data onto the eigenbases will result in a biased set of expansion coefficients. Everson and Sirovich [12] have, however, shown that, when we know how the input data are masked or censored, we can correct for the nonorthogonality of the eigenbases. Following Connolly and Szalay [9], we consider an observed spectrum, $\boldsymbol{x}^o$, as the combination of the true spectrum (i.e., without gaps), $\boldsymbol{x}$, and a wavelength-dependent weight, $\boldsymbol{w}$. This weight is zero where

data are missing and $1/\boldsymbol{\sigma}^2$ for the remaining spectral range (with $\boldsymbol{\sigma}^2$ the variance of the spectral elements). Minimizing the quadratic deviation between the original spectrum, $\boldsymbol{x}^o$, and its truncated reconstruction, $\sum_i \theta_i \boldsymbol{e}_i$ and solving for θ_i gives

$$\sum_k \theta_i \boldsymbol{w}(k)\boldsymbol{e}_i(k)\boldsymbol{e}_j(k) = \sum_k \boldsymbol{w}(k)\boldsymbol{x}^o(k)\boldsymbol{e}_j(k), \tag{7.21}$$

where $\sum_k$ represents the sum over the length of the vector $\boldsymbol{x}(k)$ (i.e., over wavelength for the case of the spectra). If we define $M_{ij} = \sum_k \boldsymbol{w}(k)\boldsymbol{e}_i(k)\boldsymbol{e}_j(k)$ and $F_i = \sum_k \boldsymbol{w}(k)\boldsymbol{x}^o(k)\boldsymbol{e}_i(k)$ then this simplifies to

$$\theta_i = \sum_j M_{ij}^{-1} F_j, \tag{7.22}$$

where F_j represent the coefficients derived from the gappy data and M_{ij}^{-1} expresses how correlated the eigenvectors are over the missing regions.

Figure 7.7 shows the reconstruction of missing data using the PCA components. The gray regions represent intervals in wavelength space where the spectra are censored (the underlying data values within these censored regions are shown by the black line). The gray lines represent the reconstruction of these spectral regions using the eigenbases. An estimate of the uncertainty on the reconstruction coefficients is given by

$$\mathrm{Cov}(\theta_i, \theta_j) = M_{ij}^{-1}. \tag{7.23}$$

The accuracy of this reconstruction will depend on the distribution of the gaps within the data vector (though this is reflected in $\mathrm{Cov}(\theta_i, \theta_j)$). For an uncorrelated set of gaps, reconstruction with the number of sampled points, $N_{\mathrm{samples}} > r$, is possible. This observation is at the heart of the fields of lossy compression and compressed sensing.

7.3.4. Scaling to Large Data Sets

There are a number of limitations of PCA that can make it impractical for application to very large data sets. Principal to this are the computational and memory requirements of the SVD, which scale as $\mathcal{O}(D^3)$ and $\mathcal{O}(2 \times D \times D)$, respectively. In §7.3.1 we derived the PCA by applying an SVD to the covariance and correlation matrices of the data X. Thus, the computational requirements of the SVD are set by the rank of the data matrix, X, with the covariance matrix the preferred route if $K < N$ and the correlation matrix if $K > N$. Given the symmetric nature of both the covariance and correlation matrix, eigenvalue decompositions (EVD) are often more efficient than SVD approaches.

Even given these optimizations, with data sets exceeding the size of the memory available per core, applications of PCA can be very computationally challenging. This is particularly the case for real-world applications when the correction techniques for missing data are iterative in nature. One approach to address these limitations is to make use of online algorithms for an iterative calculation of the mean. As shown in

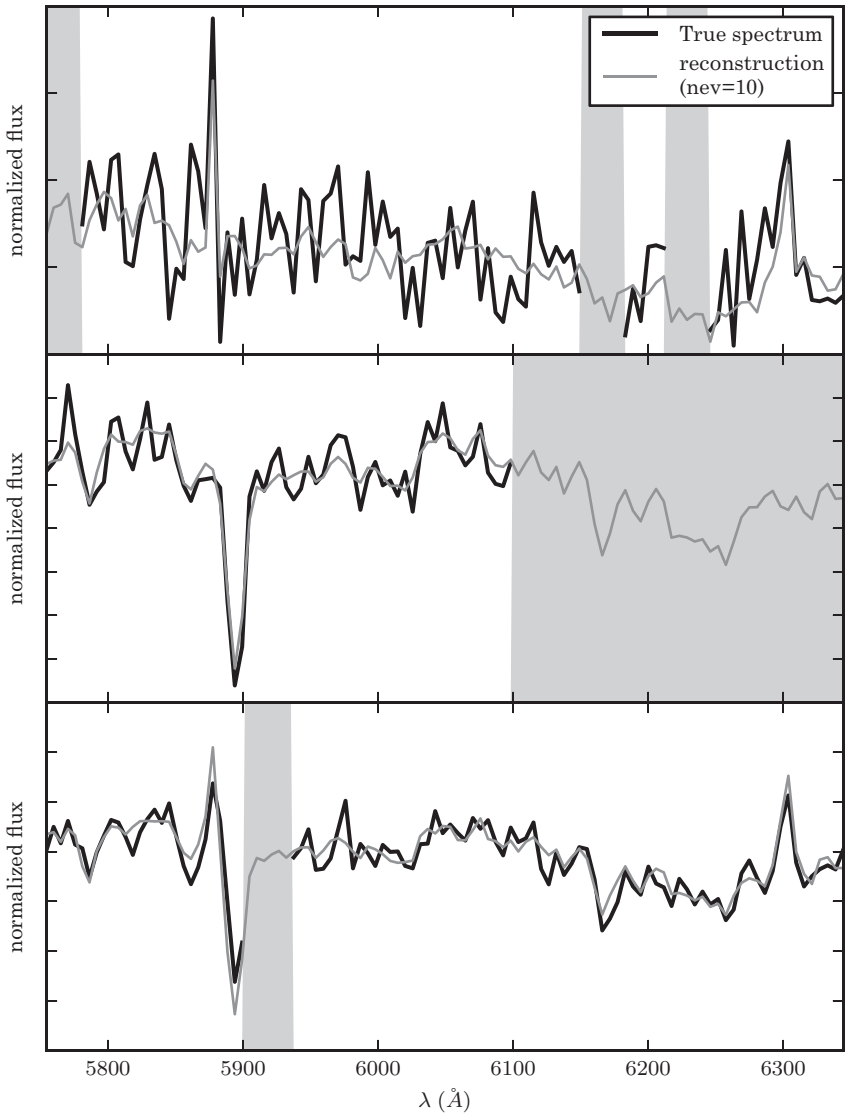


Figure 7.7. The principal component vectors defined for the SDSS spectra can be used to interpolate across or reconstruct missing data. Examples of three masked spectral regions are shown comparing the reconstruction of the input spectrum (black line) using the mean and the first ten eigenspectra (gray line) The gray bands represent the masked region of the spectrum.

[5], the sample covariance matrix can be defined as

$$C = \gamma C_{\text{prev}} + (1 - \gamma) x^T x \tag{7.24}$$

$$\sim \gamma Y_p D_p Y_p^T + (1 - \gamma) x^T x, \tag{7.25}$$

where C_{prev} is the covariance matrix derived from a previous iteration, x is the new observation, Y_p are the first p eigenvectors of the previous covariance matrix, D_p

are the associated eigenvalues, and γ is a weight parameter that can scale with step size.

For each new observation the covariance matrix is updated, and the PCA components based on this matrix enable the identification and filtering of outliers within a data set. Assuming that the number of required eigenvectors, p, is small, the cost of periodically reevaluating the SVD of the updated covariance matrix is a cheap operation.

7.4. Nonnegative Matrix Factorization

One of the challenges in interpreting PCA bases comes from the fact that the eigenvectors are defined relative to the mean data vector. This results in principal components that can be positive or negative. In contrast, for many physical systems we have a priori knowledge that a data vector can be represented by a linear sum of positive components. For example, in the case of the SDSS spectra, a galaxy spectrum can be assumed to be a linear sum of stellar components (consistent with the general form of the eigenspectra seen in figure 7.4).

Nonnegative matrix factorization (NMF) applies an additional constraint on the components that comprise the data matrix X; see [23]. It assumes that any data matrix can be factored into two matrices, W and Y, such that

$$X = WY, \tag{7.26}$$

where both W and Y are nonnegative (i.e., all elements in these matrices are nonnegative). WY is, therefore, an approximation of X. By minimizing the reconstruction error $||(X - WY)^2||$, it can be shown that nonnegative bases can be derived using a simple update rule,

$$W_{ki} = W_{ki} \frac{[XY^T]_{ki}}{[WYY^T]_{ki}}, \tag{7.27}$$

$$Y_{in} = Y_{in} \frac{[W^T X]_{in}}{[W^T WY]_{in}}, \tag{7.28}$$

where n, k, and i denote the wavelength, spectrum and template indices, respectively [23]. This iterative process does not guarantee nonlocal minima (as with many iterative machine learning approaches such as K-means and EM). With random initialization and cross-validation the solutions for the NMF bases are, however, often appropriate.

The central panel of figure 7.7 shows the results of NMF applied to the spectra used in the PCA analysis of §7.3.2. The components derived by NMF are broadly consistent with those from PCA but with a different ordering of the basis functions. Given that both applications assume a linear transform between X and Y this might not be that surprising. For the case of NMF, the assumption is that each spectrum can be represented by a smaller number of nonnegative components than the underlying dimensionality of the data. The number of components are, therefore, defined prior

to the generation of the NMF and can be derived through cross-validation techniques described in §8.11.

Projecting onto the NMF bases is undertaken in a similar manner to eq. 7.27 except that, in this case, the individual components are held fixed; see [3].

Scikit-learn contains an implementation of NMF. The basic usage is as follows:

```
import numpy as np
from sklearn.decomposition import NMF

X = np.random.random((100, 3))  # 100 points in 3
    # dims, all positive
nmf = NMF(n_components=3)  # setting n_components is
      # optional
nmf.fit(X)
proj = nmf.transform(X)  # project to 3 dimensions

comp = nmf.components_  # 3 x 10 array of components
err = nmf.reconstruction_err_  # how well 3
      # components captures data
```

There are many options to tune this procedure: for more information, refer to the Scikit-learn documentation.

7.5. Manifold Learning

PCA, NMF, and other linear dimensionality techniques are powerful ways to reduce the size of a data set for visualization, compression, or to aid in classification and regression. Real-world data sets, however, can have very nonlinear features which are hard to capture with a simple linear basis. For example, as we noted before, while quiescent galaxies can be well described by relatively few principal components, emission-line galaxies and quasars can require up to $\sim$ 30 linear components to completely characterize. These emission lines are nonlinear features of the spectra, and nonlinear methods are required to project that information onto fewer dimensions.

Manifold learning comprises a set of recent techniques which aim to accomplish this sort of nonlinear dimensionality reduction. A classic test case for this is the *S-curve* data set, shown in figure 7.8. This is a three-dimensional space, but the points are drawn from a two-dimensional manifold which is embedded in that space. Principal component analysis cannot capture this intrinsic information (see the upper-right panel of figure 7.8). There is no linear projection in which distant parts of the nonlinear manifold do not overlap. Manifold learning techniques, on the other hand, do allow this surface to be unwrapped or unfolded so that the underlying structure becomes clear.

In light of this simple example, one may wonder what can be gained from such an algorithm. While projecting from three to two dimensions is a neat trick, these

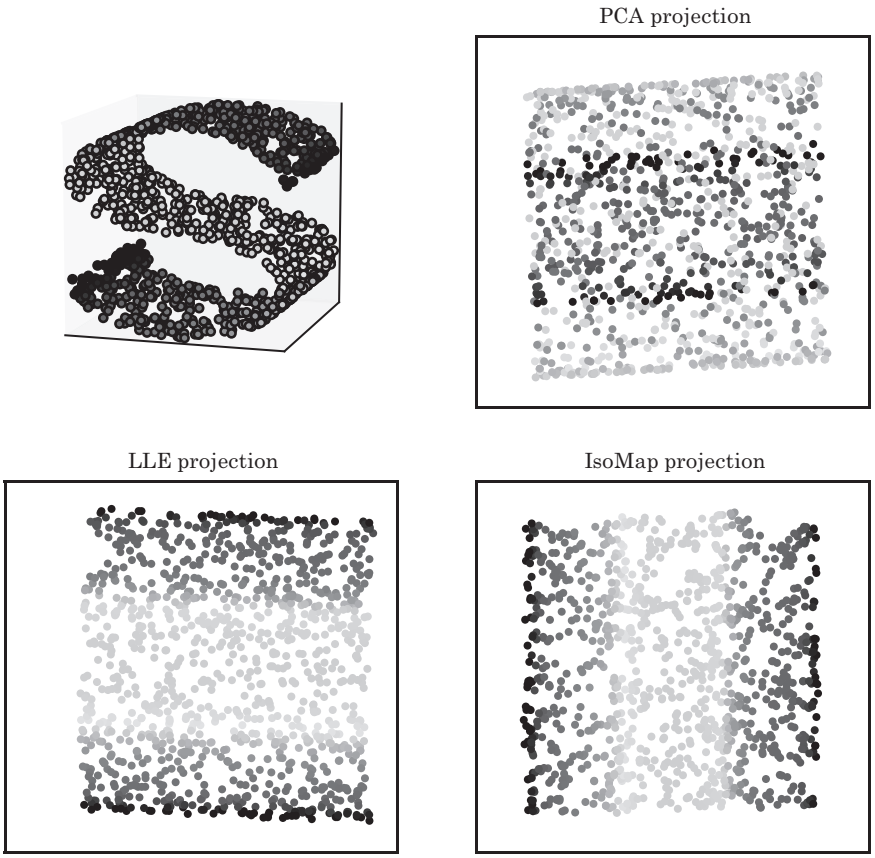


Figure 7.8. A comparison of PCA and manifold learning. The top-left panel shows an example S-shaped data set (a two-dimensional manifold in a three-dimensional space). PCA identifies three principal components within the data. Projection onto the first two PCA components results in a mixing of the colors along the manifold. Manifold learning (LLE and IsoMap) preserves the local structure when projecting the data, preventing the mixing of the colors. See color plate 5.

algorithms become very powerful when working with data like galaxy and quasar spectra, which lie in up to 4000 dimensions. Vanderplas and Connolly [32] first applied manifold learning techniques to galaxy spectra, and found that as few as two nonlinear components are sufficient to recover information which required dozens of components in a linear projection.

There are a variety of manifold learning techniques and variants available. Here we will discuss the two most popular: locally linear embedding (LLE) and IsoMap, short for isometric mapping.

7.5.1. Locally Linear Embedding

Locally linear embedding [29] is an unsupervised learning algorithm which attempts to embed high-dimensional data in a lower-dimensional space while preserving the geometry of local neighborhoods of each point. These local neighborhoods

are determined by the relation of each point to its k nearest neighbors. The LLE algorithm consists of two steps: first, for each point, a set of weights is derived which best reconstruct the point from its k nearest neighbors. These weights encode the local geometry of each neighborhood. Second, with these weights held fixed, a new lower-dimensional data set is found which maintains the neighborhood relationships described by these weights.

Let us be more specific. Let X be an $N \times K$ matrix representing N points in K dimensions. We seek an $N \times N$ weight matrix W which minimizes the reconstruction error

$$\mathcal{E}_1(W) = |X - WX|^2 \tag{7.29}$$

subject to certain constraints on W which we will mention shortly.

Let us first examine this equation and think about what it means. With some added notation, we can write this in a way that is a bit more intuitive. Each point in the data set represented by X is a K-dimensional row vector. We will denote the ith row vector by $\mathbf{x_i}$. Each point also has a corresponding weight vector given by the ith row of the weight matrix W. The portion of the reconstruction error associated with this single point can be written

$$\mathcal{E}_1(W) = \sum_{i=1}^{N} \left| \mathbf{x_i} - \sum_{j=1}^{N} W_{ij} \mathbf{x_j} \right|^2 . \tag{7.30}$$

What does it mean to minimize this equation with respect to the weights W? What we are doing is finding the linear combination of points in the data set which best reconstructs each point from the others. This is, essentially, finding the hyperplane that best describes the local surface at each point within the data set. Each row of the weight matrix W gives a set of weights for the corresponding point. As written above, the expression can be trivially minimized by setting $W = I$, the identity matrix. In this case, $WX = X$ and $\mathcal{E}_1(W) = 0$. To prevent this simplistic solution, we can constrain the problem such that the diagonal $W_{ii} = 0$ for all i. This constraint leads to a much more interesting solution. In this case the matrix W would in some sense encode the *global* geometric properties of the data set: how each point relates to all the others.

The key insight of LLE is to take this one step further, and constrain all $W_{ij} = 0$ *except* when point j is one of the k nearest neighbors of point i. With this constraint in place, the resulting matrix W has some interesting properties. First, W becomes very sparse for $k \ll N$. Out of the N^2 entries in W, only Nk are nonzero. Second, the rows of W encode the *local* properties of the data set: how each point relates to its nearest neighbors. W as a whole encodes the aggregate of these local properties, and thus contains global information about the geometry of the data set, viewed through the lens of connected local neighborhoods.

The second step of LLE mirrors the first step, but instead seeks an $N \times d$ matrix Y, where $d < D$ is the dimension of the embedded manifold. Y is found by minimizing the quantity

$$\mathcal{E}_2(Y) = |Y - WY|^2 , \tag{7.31}$$

where this time W is kept fixed. The symmetry between eqs. 7.29 and 7.31 is clear. Because of this symmetry and the constraints put on W, local neighborhoods in the

low-dimensional embedding, Y, will reflect the properties of corresponding local neighborhoods in X. This is the sense in which the embedding Y is a good nonlinear representation of X.

Algorithmically, the solutions to eqs. 7.29 and 7.31 can be obtained analytically using efficient linear algebra techniques. The details are available in the literature [29, 32], but we will summarize the results here. Step 1 requires a nearest-neighbor search (see §2.5.2), followed by a least-squares solution to the corresponding row of the weight matrix W. Step 2 requires an eigenvalue decomposition of the matrix $C_W \equiv (I - W)^T(I - W)$, which is an $N \times N$ sparse matrix, where N is the number of points in the data set. Algorithms for direct eigenvalue decomposition scale as $\mathcal{O}(N^3)$, so this calculation can become prohibitively expensive as N grows large. Iterative methods can improve on this: Arnoldi decomposition (related to the Lanczos method) allows a few extremal eigenvalues of a sparse matrix to be found relatively efficiently. A well-tested tool for Arnoldi decomposition is the Fortran package ARPACK [24]. A full Python wrapper for ARPACK is available in the functions `scipy.sparse.linalg.eigsh` (for symmetric matrices) and `scipy.sparse.linalg.eigs` (for asymmetric matrices) in SciPy version 0.10 and greater. These tools are used in the manifold learning routines available in Scikit-learn: see below.

In the astronomical literature there are cases where LLE has been applied to data as diverse as galaxy spectra [32], stellar spectra [10], and photometric light curves [27]. In the case of spectra, the authors showed that the LLE projection results in a low-dimensional representation of the spectral information, while maintaining physically important nonlinear properties of the sample (see figure 7.9). In the case of light curves, the LLE has been shown useful in aiding automated classification of observed objects via the projection of high-dimensional data onto a one-dimensional nonlinear sequence in the parameter space.

Scikit-learn has a routine to perform LLE, which uses a fast tree for neighbor search, and ARPACK for a fast solution of the global optimization in the second step of the algorithm. It can be used as follows:

```
import numpy as np
from sklearn.manifold import LocallyLinearEmbedding

X = np.random.normal(size=(1000, 2))
    # 100 pts in 2 dims
R = np.random.random((2, 10))  # projection matrix
X = np.dot(X, R)  # now a 2D linear manifold in 10D
    # space
k = 5  # number of neighbors used in the fit
n = 2  # number of dimensions in the fit
lle = LocallyLinearEmbedding(k, n)
lle.fit(X)
proj = lle.transform(X)  # 100 x 2 projection of data
```

There are many options available for the LLE computation, including more robust variants of the algorithm. For details, see the Scikit-learn documentation, or the code associated with the LLE figures in this chapter.

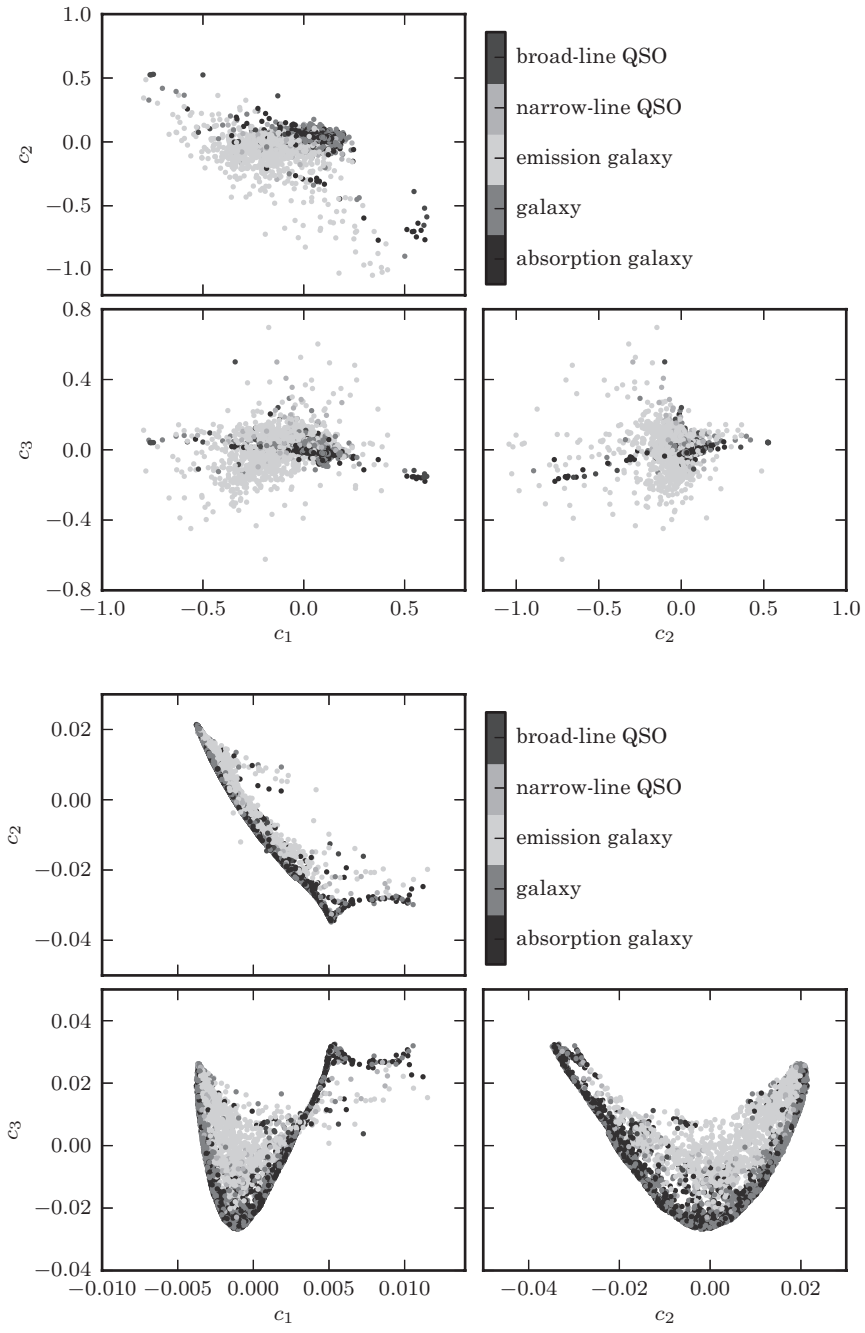


Figure 7.9. A comparison of the classification of quiescent galaxies and sources with strong line emission using LLE and PCA. The top panel shows the segregation of galaxy types as a function of the first three PCA components. The lower panel shows the segregation using the first three LLE dimensions. The preservation of locality in LLE enables nonlinear features within a spectrum (e.g., variation in the width of an emission line) to be captured with fewer components. This results in better segregation of spectral types with fewer dimensions. See color plate 6.

7.5.2. IsoMap

IsoMap [30], short for isometric mapping, is another manifold learning method which, interestingly, was introduced in the same issue of *Science* in 2000 as was LLE. IsoMap is based on a multidimensional scaling (MDS) framework. Classical MDS is a method to reconstruct a data set from a matrix of pairwise distances (for a detailed discussion of MDS see [4]).

If one has a data set represented by an $N \times K$ matrix X, then one can trivially compute an $N \times N$ distance matrix D_X such that $[D_X]_{ij}$ contains the distance between points i and j. Classical MDS seeks to reverse this operation: given a distance matrix D_X, MDS discovers a new data set Y which minimizes the error

$$\mathcal{E}_{XY} = |\tau(D_X) - \tau(D_Y)|^2, \tag{7.32}$$

where τ is an operator with a form chosen to simplify the analytic form of the solution. In metric MDS the operator τ is given by

$$\tau(D) = \frac{HSH}{2}, \tag{7.33}$$

where S is the matrix of square distances $S_{ij} = D_{ij}^2$, and H is the "centering matrix" $H_{ij} = \delta_{ij} - 1/N$. This choice of τ is convenient because it can then be shown that the optimal embedding Y is identical to the top D eigenvectors of the matrix $\tau(D_X)$ (for a derivation of this property see [26]).

The key insight of IsoMap is that we can use this metric MDS framework to derive a nonlinear embedding by constructing a suitable stand-in for the distance matrix D_X. IsoMap recovers nonlinear structure by approximating geodesic curves which lie within the embedded manifold, and computing the distances between each point in the data set along these geodesic curves. To accomplish this, the IsoMap algorithm creates a connected graph G representing the data, where G_{ij} is the distance between point i and point j if points i and j are neighbors, and $G_{ij} = 0$ otherwise. Next, the algorithm constructs a matrix D'_X such that $[D'_X]_{ij}$ contains the length of the shortest path between point i and j traversing the graph G. Using this distance matrix, the optimal d-dimensional embedding is found using the MDS algorithm discussed above.

IsoMap has a computational cost similar to that of LLE if clever algorithms are used. The first step (nearest-neighbor search) and final step (eigendecomposition of an $N \times N$ matrix) are similar to those of LLE. IsoMap has one additional hurdle however: the computation of the pairwise shortest paths on an order-N sparse graph G. A brute-force approach to this sort of problem is prohibitively expensive: for each point, one would have to test every combination of paths, leading to a total computation time of $\mathcal{O}(N^2 k^N)$. There are known algorithms which improve on this: the Floyd–Warshall algorithm [13] accomplishes this in $\mathcal{O}(N^3)$, while the Dijkstra algorithm using Fibonacci heaps [14] accomplishes this in $\mathcal{O}(N^2(k + \log N))$: a significant improvement over brute force.

Scikit-learn has a fast implementation of the IsoMap algorithm, using either the Floyd–Warshall algorithm or Dijkstra's algorithm for shortest-path search. The neighbor search is implemented with a fast tree search, and the final eigenanalysis is implemented using the Scikit-learn ARPACK wrapper. It can be used as follows:

```
import numpy as np
from sklearn.manifold import Isomap

X = np.random.normal(size=(1000, 2))
    # 1000 pts in 2 dims
R = np.random.random((2, 10))  # projection matrix
X = np.dot(X, R)  # X is now a 2D manifold in
    # 10D space
k = 5  # number of neighbors used in the fit
n = 2  # number of dimensions in the fit
iso = Isomap(k, n)
iso.fit(X)
proj = iso.transform(X)  # 1000 x 2 projection of
       # data
```

For more details, see the documentation of Scikit-learn or the source code of the IsoMap figures in this chapter.

7.5.3. Weaknesses of Manifold Learning

Manifold learning is a powerful tool to recover low-dimensional nonlinear projections of high-dimensional data. Nevertheless, there are a few weaknesses that prevent it from being used as widely as techniques like PCA:

Noisy and gappy data: Manifold learning techniques are in general not well suited to fitting data plagued by noise or gaps. To see why, imagine that a point in the data set shown in figure 7.8 is located at $(x, y) = (0, 0)$, but not well constrained in the z direction. In this case, there are three perfectly reasonable options for the missing z coordinate: the point could lie on the bottom of the "S", in the middle of the "S", or on the top of the "S". For this reason, manifold learning methods will be fundamentally limited in the case of missing data. One may imagine, however, an iterative approach which would construct a (perhaps multimodal) Bayesian constraint on the missing values. This would be an interesting direction for algorithmic research, but such a solution has not yet been demonstrated.

Tuning parameters: In general, the nonlinear projection obtained using these techniques depends highly on the set of nearest neighbors used for each point. One may select the k neighbors of each point, use all neighbors within a radius r of each point, or choose some more sophisticated technique. There is currently no solid recommendation in the literature for choosing the optimal set of neighbors for a given embedding: the optimal choice will depend highly on the local density of each point, as well as the curvature of the manifold at each point. Once again, one may

imagine an iterative approach to optimizing the selection of neighbors based on these insights. This also could be an interesting direction for research.

Dimensionality: One nice feature of PCA is that the dimensionality of the data set can be estimated, to some extent, from the eigenvalues associated with the projected dimensions. In manifold learning, there is no such clean mapping. In fact, we have no guarantee that the embedded manifold is unidimensional! One can easily imagine a situation where a data set is drawn from a two-dimensional manifold in one region, and from a three-dimensional manifold in an adjacent region. Thus in a manifold learning setting, the choice of output dimensionality is a free parameter. In practice, either $d = 1$, $d = 2$, or $d = 3$ is often chosen, for the simple reason that it leads to a projection which is easy to visualize! Some attempts have been made to use local variance of a data set to arrive at an estimate of the dimensionality (see, e.g., [11]) but this works only marginally well in practice.

Sensitivity to outliers: Another weakness of manifold learning is its sensitivity to outliers. In particular, even a single outlier between different regions of the manifold can act to "short-circuit" the manifold so that the algorithm cannot find the correct underlying embedding.

Reconstruction from the manifold: Because manifold learning methods generally do not provide a set of basis functions, any mapping from the embedded space to the higher-dimensional input space must be accomplished through a reconstruction based on the location of nearest neighbors. This means that a projection derived from these methods cannot be used to compress data in a way that is analogous to PCA. The full input data set and the full projected data must be accessed in order to map new points between the two spaces.

With these weaknesses in mind, manifold learning techniques can still be used successfully to analyze and visualize large astronomical data sets. LLE has been applied successfully to several classes of data, both as a classification technique and an outlier detection technique [10, 27, 32]. These methods are ripe for exploration of the high-dimensional data sets available from future large astronomical surveys.

7.6. Independent Component Analysis and Projection Pursuit

Independent component analysis (ICA) [8] is a computational technique that has become popular in the biomedical signal processing community to solve what has often been referred to as the "cocktail party problem"; see [7]. In this problem, there are multiple microphones situated through out a room containing N people. Each microphone picks up a linear combination of the N voices. The goal of ICA is to use the concept of statistical independence to isolate (or unmix) the individual signals. In the context of astronomical problems we will consider the application of ICA to the series of galaxy spectra used for PCA (see §7.3.2). In this example, each galaxy spectrum is considered as the microphone picking up a linear combination of input signals from individual stars and HII regions.

Each spectrum, $x_i(k)$, can now be described by

$$x_1(k) = a_{11}s_1(k) + a_{12}s_2(k) + a_{13}s_3(k) + \cdots, \tag{7.34}$$

$$x_2(k) = a_{21}s_1(k) + a_{22}s_2(k) + a_{23}s_3(k) + \cdots, \tag{7.35}$$

$$x_3(k) = a_{31}s_1(k) + a_{32}s_2(k) + a_{33}s_3(k) + \cdots, \tag{7.36}$$

where $s_i(k)$ are the individual stellar spectra and a_{ij} the appropriate mixing amplitudes. In matrix format we can write this as

$$X = AS, \tag{7.37}$$

where X and S are matrices for the set of input spectra and stellar spectra, respectively. Extracting these signal spectra is equivalent to estimating the appropriate weight matrix, W, such that

$$S = WX. \tag{7.38}$$

The principle that underlies ICA comes from the observation that the input signals, $s_i(k)$, should be statistically independent. Two random variables are considered statistically independent if their joint probability distribution, $f(x, y)$, can be fully described by a combination of their marginalized probabilities, that is,

$$f(x^p, y^q) = f(x^p)f(y^q), \tag{7.39}$$

where p and q represent arbitrary higher-order moments of the probability distributions. For the case of PCA, $p = q = 1$ and the statement of independence simplifies to the weaker condition of uncorrelated data (see §7.3.1 on the derivation of PCA).

In most implementations of ICA algorithms the requirement for statistical independence is expressed in terms of the non-Gaussianity of the probability distributions. The rationale for this is that the sum of any two independent random variables will always be more Gaussian than either of the individual random variables (i.e., from the central limit theorem). This would mean that, for the case of the stellar components that make up a galaxy spectrum, if we identify an unmixing matrix, W, that maximizes the non-Gaussianity of the distributions, then we would be identifying the input signals. Definitions of non-Gaussianity range from the use of the kurtosis of a distribution (see §5.2.2), the negentropy (the negative of the entropy of a distribution), and mutual information.

Related to ICA is projection pursuit [15, 16]. Both techniques seek interesting directions within multivariate data sets. One difference is that in projection pursuit these directions are identified one at a time, while for ICA the search for these signals can be undertaken simultaneously. For each case, the definition of "interesting" is often expressed in terms of how non-Gaussian the distributions are after projections. Projection Pursuit can be considered as a subset of ICA.

Scikit-learn has an implementation of ICA based on the FastICA algorithm [18]. It can be used as follows:

```
import numpy as np
from sklearn.decomposition import FastICA

X = np.random.normal(size=(100, 2))
    # 100 pts in 2 dims
R = np.random.random((2, 5))  # mixing matrix
X = np.dot(X, R)  # X is now 2D data in 5D space
ica = FastICA(2)  # fit two components
ica.fit(X)
proj = ica.transform(X) # 100 x 2 projection of data

comp = ica.components_  # the 2 x 5 matrix of indep.
       # components
sources = ica.sources_  # the 100 x 2 matrix of
          # sources
```

There are several options to fine-tune the algorithm; for details refer to the Scikit-learn documentation.

7.6.1. The Application of ICA to Astronomical Data

In figure 7.1 we showed the set of spectra used to test PCA. From the eigenspectra and their associated eigenvalues it was apparent that each spectrum comprises a linear combination of basis functions whose shapes are broadly consistent with the spectral properties of individual stellar types (O, A, and G stars). In the middle panel of figure 7.4 we apply ICA to these same spectra to define the independent components. As with PCA, preprocessing of the input data is an important component of any ICA application. For each data set the mean vector is removed to center the data. Whitening of the distributions (where the covariance matrix is diagonalized and normalized to reduce it to the identity matrix) is implemented through an eigenvalue decomposition of the covariance matrix.

The cost function employed is that of FastICA [18] which uses an analytic approximation to the negentropy of the distributions. The advantage of FastICA is that each of the independent components can be evaluated one at a time. Thus the analysis can be terminated once a sufficient number of components has been identified.

Comparison of the components derived from PCA, NMF and ICA in figure 7.4 shows that each of these decompositions produces a set of basis functions that are broadly similar (including both continuum and line emission). The ordering of the importance of each component is dependent on technique: in the case of ICA, finding a subset of ICA components is not the same as finding all ICA components (see [17]), which means that the a priori assumption of the number of underlying components will affect the form of the resulting components. This choice of dimensionality is a problem common to all dimensionality techniques (from LLE to the truncation of

PCA components). As with many multivariate applications, as the size of the mixing matrix grows, computational complexity often makes it impractical to calculate the weight matrix W directly. Reduction in the complexity of the input signals through the use of PCA (either to filter the data or to project the data onto these basis functions) is often applied to ICA applications.

7.7. Which Dimensionality Reduction Technique Should I Use?

In chapter 6 we introduced the concept of defining four axes against which we can compare the techniques described in each chapter. Using the axes of "accuracy," "interpretability," "simplicity," and "speed" (see §6.6 for a description of these terms) we provide a rough assessment of each of the methods considered in this chapter.

What are the most *accurate* dimensionality reduction methods? First we must think about how we want to define the notion of accuracy for the task of dimension reduction. In some sense the different directions one can take here are what distinguishes the various methods. One clear notion of this is in terms of reconstruction error—some function of the difference between the original data matrix and the data matrix reconstructed using the desired number of components. When all possible components are used, every dimensionality reduction method, in principle, is an exact reconstruction (zero error), so the question becomes some function of the number of components—for example, which method tends to yield the most faithful reconstruction after K components. PCA is designed to provide the best square reconstruction error for any given K. LLE minimizes square reconstruction error, but in a nonlinear fashion. NMF also minimizes a notion of reconstruction error, under nonnegativity constraints. Other approaches, such as the multidimensional scaling family of methods ([4, 22]), of which IsoMap is a modern descendant and variant, attempt to minimize the difference between each pairwise distance in the original space and its counterpart in the reconstructed space. In other words, once one has made a choice of which notion of accuracy is desired, there is generally a method which directly maximizes that notion of accuracy. Generally speaking, for a fixed notion of error, a nonlinear model should better minimize error than one which is constrained to be linear (i.e., manifold learning techniques should provide more compact and accurate representations of the data). Additional constraints, such as nonnegativity, only reduce the flexibility the method has to fit the data, making, for example, NMF generally worse in the sense of reconstruction error than PCA. ICA can also be seen in the light of adding additional constraints, making it less aggressive in terms of reconstruction error.

NMF-type approaches do, however, have a significant performance advantage over PCA, ICA, and manifold learning for the case of low signal-to-noise data due to the fact that they have been expanded to account for heteroscedastic uncertainties; see [31]. NMF and its variants can model the *measured* uncertainties within the data (rather than assuming that the variance of the ensemble of data is representative of these uncertainties). This feature, at least in principle, provides a more accurate set of derived basis functions.

What are the most *interpretable* dimensionality reduction methods? In the context of dimensionality reduction methods, interpretability generally means the extent to which we can identify the meaning of the directions found by the method. For data sets where all values are known to be nonnegative, as in the case of spectra or image pixel brightnesses, NMF tends to yield more sensible components than PCA. This makes sense because the constraints ensure that reconstructions obtained from NMF are themselves valid spectra or images, which might not be true otherwise. Interpretability in terms of understanding each component through the original dimensions it captures can be had to some extent with linear models like PCA, ICA, and NMF, but is lost in the nonlinear manifold learning approaches like LLE and IsoMap (which do not provide the principal directions, but rather the positions of objects within a lower-dimensional space).

A second, and significant, advantage of PCA over the other techniques is the ability to estimate the importance of each principal component (in terms of its contribution to the variance). This enables simple, though admittedly ad hoc (see §7.3.2), criteria to be applied when truncating the projection of data onto a set of basis functions.

What are the most *scalable* dimensionality reduction methods? Faster algorithms can be obtained for PCA in part by focusing on the fact that only the top K components are typically needed. Approximate algorithms for SVD-like computations (where only parts of the SVD are obtained) based on sampling are available, as well as algorithms using similar ideas that yield full SVDs. Online algorithms can be fairly effective for SVD-like computations. ICA can be approached a number of ways, for example using the FastICA algorithm [18]. Fast optimization methods for NMF are discussed in [23]. In general, linear methods for dimensionality reduction are usually relatively tractable, even for high N. For nonlinear methods such as LLE and IsoMap, the most expensive step is typically the first one, an $\mathcal{O}(N^2)$ all-nearest-neighbor computation to find the K neighbors for each point. Fast algorithms for such problems, including one that is provably $\mathcal{O}(N)$, are discussed in §2.4 (see also WSAS). The second step is typically a kind of SVD computation. Overall LLE is $\mathcal{O}(N^2)$ and IsoMap is $\mathcal{O}(N^3)$, making them intractable on large data sets without algorithmic intervention.

What are the *simplest* dimensionality reduction methods? All dimensionality reduction methods are fiddly in the sense that they all require, to some extent, the selection of the number of components/dimensions to which the data should be reduced. PCA's objective is convex, meaning that there is no need for multiple random restarts, making it usable essentially right out of the box. This is not the case for ICA and NMF, which have nonconvex objectives. Manifold learning methods like LLE and IsoMap have convex objectives, but require careful evaluation of the parameter k which defines the number of nearest neighbors.

As mentioned in §6.6, selecting parameters can be done automatically in principle for any machine learning method for which cross-validation (see §8.11) can be applied, including unsupervised methods as well as supervised ones. Cross-validation can be applied whenever the model can be used to make predictions (in this case "predictions" means coordinates in the new low-dimensional space) on test data, that is, data not included in the training data. While PCA, NMF, and ICA can

TABLE 7.1.
Summary of the practical properties of the main dimensionality reduction techniques.

Method	Accuracy	Interpretability	Simplicity	Speed
Principal component analysis	H	H	H	H
Locally linear embedding	H	M	H	M
Nonnegative matrix factorization	H	H	M	M
Independent component analysis	M	M	L	L

be applied to such "out-of-sample" data, LLE and IsoMap effectively require, in order to get predictions, that test data be added to the existing training set and the whole model retrained. This requirement precludes the use of cross-validation to select k automatically and makes the choice of k a subjective exercise.

Other considerations, and taste. As we noted earlier in §7.5.3, other considerations can include robustness to outliers and missing values. Many approaches have been taken to making robust versions of PCA, since PCA, as a maximum-likelihood-based model, is sensitive to outliers. Manifold learning methods can also be sensitive to outliers. PCA, ICA, and NMF can be made to work with missing values in a number of ways: we saw one example for PCA in §7.3.3. There is no clear way to handle missing values with manifold learning methods, as they are based on the idea of distances, and it is not clear how to approach the problem of computing distances with missing values. Regarding taste, if we consider these criteria as a whole the simplest and most useful technique is principal component analysis (as has been recognized by the astronomical community with over 300 refereed astronomical publications between 2000 and 2010 that mention the use of PCA). Beyond PCA, NMF has the advantage of mapping to many astronomical problems (i.e., for positive data and in the low signal-to-noise regime) and is particularly suitable for smaller data sets. ICA has interesting roots in information theory and signal processing, and manifold learning has roots in geometry; their adoption within the astronomical community has been limited to date.

Simple summary. Table 7.1 is a simple summary of the trade-offs along our axes of accuracy, interpretability, simplicity, and speed in dimension reduction methods, expressed in terms of high (H), medium (M), and low (L) categories.

References

[1] Abazajian, K. N., J. K. Adelman-McCarthy, M. A. Agüeros, and others (2009). The Seventh Data Release of the Sloan Digital Sky Survey. *ApJS 182*, 543–558.

[2] Bellman, R. E. (1961). *Adaptive Control Processes*. Princeton, NJ: Princeton University Press.

[3] Blanton, M. R. and S. Roweis (2007). K-corrections and filter transformations in the ultraviolet, optical, and near-infrared. *AJ 133*, 734–754.

[4] Borg, I. and P. Groenen (2005). *Modern Multidimensional Scaling: Theory and Applications*. Springer.

[5] Budavári, T., V. Wild, A. S. Szalay, L. Dobos, and C.-W. Yip (2009). Reliable eigenspectra for new generation surveys. *MNRAS 394*, 1496–1502.

[6] Cattell, R. B. (1966). The scree test for the number of factors. *Multivariate Behavioral Research 1*(2), 245–276.

[7] Cherry, E. C. (1953). Some experiments on the recognition of speech, with one and with two ears. *Acoustical Society of America Journal 25*, 975.

[8] Comon, P. (1994). Independent component analysis—a new concept? *Signal Processing 36*(3), 287–314.

[9] Connolly, A. J. and A. S. Szalay (1999). A robust classification of galaxy spectra: Dealing with noisy and incomplete data. *AJ 117*, 2052–2062.

[10] Daniel, S. F., A. Connolly, J. Schneider, J. Vanderplas, and L. Xiong (2011). Classification of stellar spectra with local linear embedding. *AJ 142*, 203.

[11] de Ridder, D. and R. P. Duin (2002). Locally linear embedding for classification. *Pattern Recognition Group Technical Report Series PH-2002-01*.

[12] Everson, R. and L. Sirovich (1995). Karhunen-Loeve procedure for gappy data. *J. Opt. Soc. Am. A 12*, 1657–1664.

[13] Floyd, R. W. (1962). Algorithm 97: Shortest path. *Commun. ACM 5*, 345.

[14] Fredman, M. L. and R. E. Tarjan (1987). Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM 34*, 596–615.

[15] Friedman, J. H. (1987). Exploratory projection pursuit. *Journal of the American Statistical Association 82*, 249–266.

[16] Friedman, J. H. and J. W. Tukey (1974). A projection pursuit algorithm for exploratory data analysis. *IEEE Transactions on Computers 23*, 881–889.

[17] Girolami, M. and C. Fyfe (1997). Negentropy and kurtosis as projection pursuit indices provide generalised ICA algorithms. In A. Cichocki and A.Back (Eds.), *NIPS-96 Blind Signal Separation Workshop*, Volume 8.

[18] Hyvaerinen, A. (1999). Fast and robust fixed-point algorithms for independent component analysis. *IEEE-NN 10*(3), 626.

[19] Jolliffe, I. T. (1986). *Principal Component Analysis*. Springer.

[20] Kaiser, H. F. (1960). The application of electronic computers to factor analysis. *Educational and Psychological Measurement 20*(1), 141–151.

[21] Karhunen, H. (1947). Uber Lineare Methoden in der Wahrscheinlichkeitsrechnung. *Ann. Acad. Sci. Fenn. Ser. A.I. 37*. See translation by I. Selin, The Rand Corp., Doc. T-131, 1960.

[22] Kruskal, J. and M. Wish (1978). *Multidimensional Scaling*. Number 11 in Quantitative Applications in the Social Sciences. SAGE Publications.

[23] Lee, D. D. and H. S. Seung (2001). Algorithms for non-negative matrix factorization. In T. K. Leen, T. G. Dietterich, and V. Tresp (Eds.), *Advances in Neural Information Processing Systems 13: Proceedings of the 2000 Conference*, Cambridge, Massachusetts, pp. 556–562. MIT Press.

[24] Lehoucq, R. B., D. C. Sorensen, and C. Yang (1997). ARPACK users guide: Solution of large scale eigenvalue problems by implicitly restarted Arnoldi methods.

[25] Loéve, M. (1963). *Probability Theory*. New York: Van Nostrand.

[26] Mardia, K. V., J. T. Kent, and J. M. Bibby (1979). *Multivariate Analysis*. Academic Press.

[27] Matijevič, G., A. Prša, J. A. Orosz, and others (2012). Kepler eclipsing binary stars. III. Classification of Kepler eclipsing binary light curves with locally linear embedding. *AJ 143*, 123.

[28] Pearson, K. (1901). On lines and planes of closest fit to systems of points in space. *The London, Edinburgh and Dublin Philosophical Magazine and Journal of Science 2*, 559–572.

[29] Roweis, S. T. and L. K. Saul (2000). Nonlinear dimensionality reduction by locally linear embedding. *Science 290*, 2323–2326.

[30] Tenenbaum, J. B., V. Silva, and J. C. Langford (2000). A global geometric framework for nonlinear dimensionality reduction. *Science 290*(5500), 2319–2323.

[31] Tsalmantza, P. and D. W. Hogg (2012). A data-driven model for spectra: Finding double redshifts in the Sloan Digital Sky Survey. *ApJ 753*, 122.

[32] Vanderplas, J. and A. Connolly (2009). Reducing the dimensionality of data: Locally linear embedding of Sloan Galaxy Spectra. *AJ 138*, 1365–1379.

[33] Yip, C. W., A. J. Connolly, A. S. Szalay, and others (2004). Distributions of galaxy spectral types in the Sloan Digital Sky Survey. *AJ 128*, 585–609.

[34] Yip, C. W., A. J. Connolly, D. E. Vanden Berk, and others (2004). Spectral classification of quasars in the Sloan Digital Sky Survey: Eigenspectra, redshift, and luminosity effects. *AJ 128*, 2603–2630.

8 Regression and Model Fitting

"Why are you trying so hard to fit in when you were born to stand out?" (Ian Wallace)

Regression is a special case of the general model fitting and selection procedures discussed in chapters 4 and 5. It can be defined as the relation between a dependent variable, y, and a set of independent variables, x, that describes the expectation value of y given x: $E[y|x]$. The purpose of obtaining a "best-fit" model ranges from scientific interest in the values of model parameters (e.g., the properties of dark energy, or of a newly discovered planet) to the predictive power of the resulting model (e.g., predicting solar activity). The usage of the word regression for this relationship dates back to Francis Galton, who discovered that the difference between a child and its parents for some characteristic is proportional to its parents' deviation from typical people in the whole population,[1] or that children "regress" toward the population mean. Therefore, modern usage of the word in a statistical context is somewhat different.

As we will describe below, regression can be formulated in a way that is very general. The solution to this generalized problem of regression is, however, quite elusive. Techniques used in regression tend, therefore, to make a number of simplifying assumptions about the nature of the data, the uncertainties of the measurements, and the complexity of the models. In the following sections we start with a general formulation for regression, list various simplified cases, and then discuss methods that can be used to address them, such as regression for linear models, kernel regression, robust regression and nonlinear regression.

8.1. Formulation of the Regression Problem

Given a multidimensional data set drawn from some pdf and the full error covariance matrix for each data point, we can attempt to infer the underlying pdf using either parametric or nonparametric models. In its most general incarnation, this is a

[1] If your parents have very high IQs, you are more likely to have a lower IQ than them, than a higher one. The expected probability distribution for your IQ if you also have a sister whose IQ exceeds your parents' IQs is left as an exercise for the reader. Hint: This is related to regression toward the mean discussed in § 4.7.1.

very hard problem to solve. Even with a restrictive assumption that the errors are Gaussian, incorporating the error covariance matrix within the posterior distribution is not trivial (cf. § 5.6.1). Furthermore, accounting for any selection function applied to the data can increase the computational complexity significantly (e.g., recall § 4.2.7 for the one-dimensional case), and non-Gaussian error behavior, if not accounted for, can produce biased results.

Regression addresses a slightly simpler problem: instead of determining the multidimensional pdf, we wish to infer the expectation value of y given x (i.e., the conditional expectation value). If we have a model for the conditional distribution (described by parameters $\boldsymbol{\theta}$) we can write this function[2] as $y = f(x|\boldsymbol{\theta})$. We refer to y as a scalar dependent variable and x as an independent vector. Here x does not need to be a random variable (e.g., x could correspond to deterministic sampling times for a time series). For a given model class (i.e., the function f can be an analytic function such as a polynomial, or a nonparametric estimator), we have k model parameters θ_p, $p = 1, \ldots, k$.

Figure 8.1 illustrates how the constraints on the model parameters, $\boldsymbol{\theta}$, respond to the observations x_i and y_i. In this example, we assume a simple straight-line model with $y_i = \theta_0 + \theta_1 x_i$. Each point provides a joint constraint on θ_0 and θ_1. If there were no uncertainties on the variables then this constraint would be a straight line in the (θ_0, θ_1) plane ($\theta_0 = y_i - \theta_1 x_i$). As the number of points is increased the best estimate of the model parameters would then be the intersection of all lines. Uncertainties within the data will transform the constraint from a line to a distribution (represented by the region shown as a gray band in figure 8.1). The best estimate of the model parameters is now given by the posterior distribution. This is simply the multiplication of the probability distributions (constraints) for all points and is shown by the error ellipses in the lower panel of figure 8.1. Measurements with upper limits (e.g., point x_4) manifest as half planes within the parameter space. Priors are also accommodated naturally within this picture as additional multiplicative constraints applied to the likelihood distribution (see § 8.2).

Computationally, the cost of this general approach to regression can be prohibitive (particularly for large data sets). In order to make the analysis tractable, we will, therefore, define several types of regression using three "classification axes":

- *Linearity.* When a parametric model is linear in all model parameters, that is, $f(x|\theta) = \sum_{p=1}^{k} \theta_p \, g_p(x)$, where functions $g_p(x)$ do not depend on any free model parameters (but can be nonlinear functions of x), regression becomes a significantly simpler problem, called linear regression. Examples of this include polynomial regression, and radial basis function regression. Regression of models that include nonlinear dependence on θ_p, such as $f(x|\theta) = \theta_1 + \theta_2 \sin(\theta_3 x)$, is called nonlinear regression.
- *Problem complexity.* A large number of independent variables increases the complexity of the error covariance matrix, and can become a limiting factor in nonlinear regression. The most common regression case found in practice is the $M = 1$ case with only a single independent variable (i.e., fitting a straight line to data). For linear models and negligible errors on the independent variables, the problem of dimensionality is not (too) important.

[2]Sometimes $f(x;\boldsymbol{\theta})$ is used instead of $f(x|\boldsymbol{\theta})$ to emphasize that here f is a function rather than pdf.

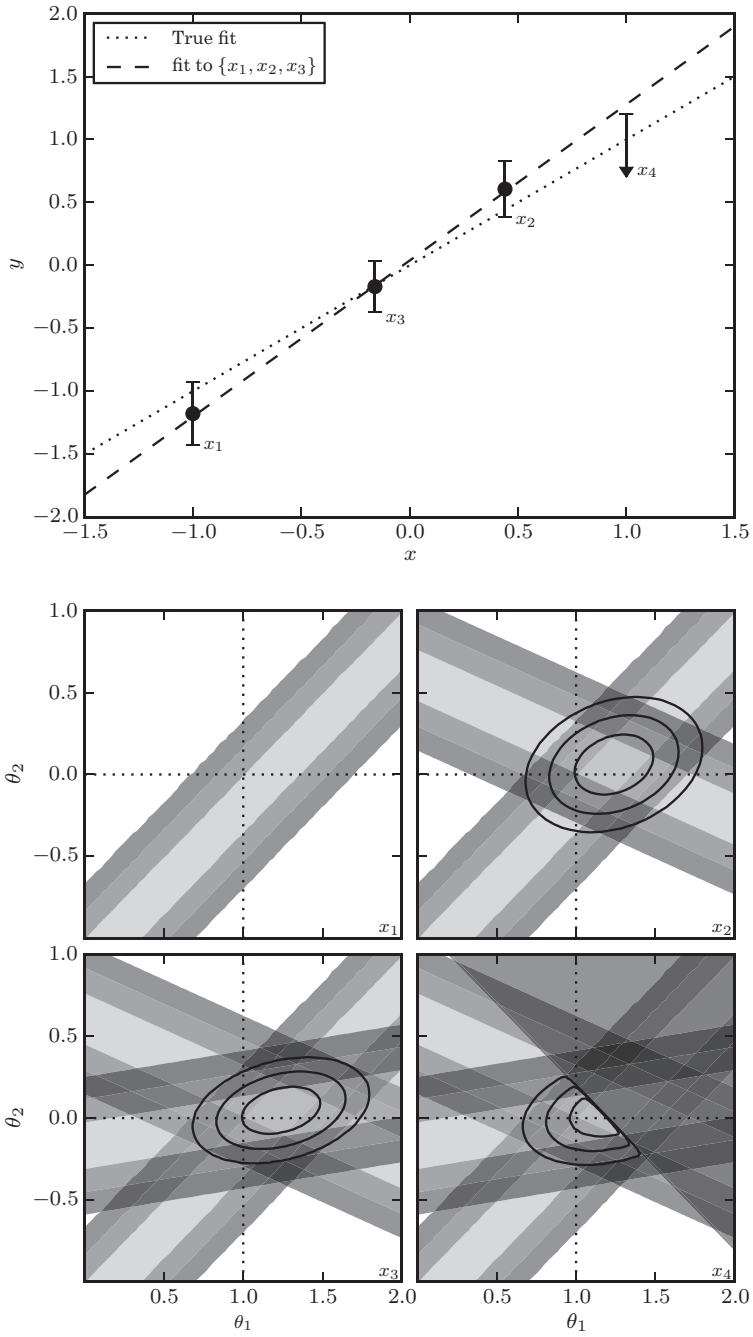


Figure 8.1. An example showing the online nature of Bayesian regression. The upper panel shows the four points used in regression, drawn from the line $y = \theta_1 x + \theta_0$ with $\theta_1 = 1$ and $\theta_0 = 0$. The lower panel shows the posterior pdf in the (θ_0, θ_1) plane as each point is added in sequence. For clarity, the implied dark regions for $\sigma > 3$ have been removed. The fourth point is an upper-limit measurement of y, and the resulting posterior cuts off half the parameter space.

- *Error behavior.* The uncertainties in the values of independent and dependent variables, and their correlations, are the primary factor that determines which regression method to use. The structure of the error covariance matrix, and deviations from Gaussian error behavior, can turn seemingly simple problems into complex computational undertakings. Here we will separately discuss the following cases:

    1. Both independent and dependent variables have negligible errors (compared to the intrinsic spread of data values); this is the simplest and most common "y vs. x" case, and can be relatively easily solved even for nonlinear models and multidimensional data.
    2. Only errors for the dependent variable (y) are important, and their distribution is Gaussian and homoscedastic (with σ either known or unknown).
    3. Errors for the dependent variable are Gaussian and known, but heteroscedastic.
    4. Errors for the dependent variable are non-Gaussian, and their behavior is known.
    5. Errors for the dependent variable are non-Gaussian, but their exact behavior is unknown.
    6. Errors for independent variables (x) are not negligible, but the full covariance matrix can be treated as Gaussian. This case is relatively straightforward when fitting a straight line, but can become cumbersome for more complex models.
    7. All variables have non-Gaussian errors. This is the hardest case and there is no ready-to-use general solution. In practice, the problem is solved on a case-by-case basis, typically using various approximations that depend on the problem specifics.

For the first four cases, when error behavior for the dependent variable is known, and errors for independent variables are negligible, we can easily use the Bayesian methodology developed in chapter 5 to write the posterior pdf for the model parameters,

$$p(\boldsymbol{\theta}|\{x_i, y_i\}, I) \propto p(\{x_i, y_i\}|\boldsymbol{\theta}, I)\, p(\boldsymbol{\theta}, I). \tag{8.1}$$

Here the information I describes the error behavior for the dependent variable. The data likelihood is the product of likelihoods for the individual points, and the latter can be expressed as

$$p(y_i|x_i, \boldsymbol{\theta}, I) = e(y_i|y), \tag{8.2}$$

where $y = f(x|\boldsymbol{\theta})$ is the adopted model class, and $e(y_i|y)$ is the probability of observing y_i given the true value (or the model prediction) y. For example, if the y error distribution is Gaussian, with the width for ith data point given by σ_i, and the errors on x are negligible, then

$$p(y_i|x_i, \boldsymbol{\theta}, I) = \frac{1}{\sigma_i\sqrt{2\pi}} \exp\left(\frac{-[y_i - f(x_i|\boldsymbol{\theta})]^2}{2\sigma_i^2}\right). \tag{8.3}$$

8.1.1. Data Sets Used in This Chapter

For regression and its application to astrophysics we focus on the relation between the redshifts of supernovas and their luminosity distance (i.e., a cosmological parametrization of the expansion of the universe [1]). To accomplish this we generate a set of synthetic supernova data assuming a cosmological model given by

$$\mu(z) = -5\log_{10}\left((1+z)\frac{c}{H_0}\int \frac{dz}{(\Omega_m(1+z)^3+\Omega_\Lambda)^{1/2}}\right), \tag{8.4}$$

where $\mu(z)$ is the distance modulus to the supernova, H_0 is the Hubble constant, Ω_m is the cosmological matter density and Ω_Λ is the energy density from a cosmological constant. For our fiducial cosmology we choose $\Omega_m = 0.3$, $\Omega_\Lambda = 0.7$ and $H_0 = 70\,\mathrm{km\,s^{-1}\,Mpc^{-1}}$, and add heteroscedastic Gaussian noise that increases linearly with redshift. The resulting $\mu(z)$ cannot be expressed as a sum of simple closed-form analytic functions, including low-order polynomials. This example addresses many of the challenges we face when working with observational data sets: we do not know the intrinsic complexity of the model (e.g., the form of dark energy), the dependent variables can have heteroscedastic uncertainties, there can be missing or incomplete data, and the dependent variables can be correlated. For the majority of techniques described in this chapter we will assume that uncertainties in the independent variables are small (relative to the range of data and relative to the dependent variables). In real-world applications we do not get to make this choice (the observations themselves define the distribution in uncertainties irrespective of the models we assume). For the supernova data, an example of such a case would be if we estimated the supernova redshifts using broadband photometry (i.e., photometric redshifts). Techniques for addressing such a case are described in § 8.8.1. We also note that this toy model data set is a simplification in that it does not account for the effect of K-corrections on the observed colors and magnitudes; see [7].

8.2. Regression for Linear Models

Given an independent variable x and a dependent variable y, we will start by considering the simplest case, a linear model with

$$y_i = \theta_0 + \theta_1 x_i + \epsilon_i. \tag{8.5}$$

Here θ_0 and θ_1 are the coefficients that describe the regression (or objective) function that we are trying to estimate (i.e., the slope and intercept for a straight line $f(x) = \theta_0 + \theta_1 x_i$), and ϵ_i represents an additive noise term.

The assumptions that underlie our linear regression model include the uncertainties on the independent variables that are considered to be negligible, and the dependent variables have known heteroscedastic uncertainties, $\epsilon_i = \mathcal{N}(0, \sigma_i)$. From eq. 8.3 we can write the data likelihood as

$$p(\{y_i\}|\{x_i\}, \boldsymbol{\theta}, I) = \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi}\sigma_i} \exp\left(\frac{-(y_i - (\theta_0 + \theta_1 x_i))^2}{2\sigma_i^2}\right). \tag{8.6}$$

For a flat or uninformative prior pdf, $p(\theta|I)$, where we have no knowledge about the distribution of the parameters θ, the posterior will be directly proportional to the likelihood function (which is also known as the error function). If we take the logarithm of the posterior then we arrive at the classic definition of regression in terms of the log-likelihood:

$$\ln(L) \equiv \ln((\boldsymbol{\theta}|\{x_i, y_i\}, I)) \propto \sum_{i=1}^{N} \left(\frac{-(y_i - (\theta_0 + \theta_1 x_i))^2}{2\sigma_i^2} \right). \tag{8.7}$$

Maximizing the log-likelihood as a function of the model parameters, $\boldsymbol{\theta}$, is achieved by minimizing the sum of the square errors. This observation dates back to the earliest applications of regression with the work of Gauss [6] and Legendre [14], when the technique was introduced as the "method of least squares."

The form of the likelihood function and the "method of least squares" optimization arises from our assumption of Gaussianity for the distribution of uncertainties in the dependent variables. Other forms for the likelihoods can be assumed (e.g., using the L_1 norm, see § 4.2.8, which actually precedes the use of the L_2 norm [2, 13], but this is usually at the cost of increased computational complexity). If it is known that measurement errors follow an exponential distribution (see § 3.3.6) instead of a Gaussian distribution, then the L_1 norm should be used instead of the L_2 norm and eq. 8.7 should be replaced by

$$\ln(L) \propto \sum_{i=1}^{N} \left(\frac{-|y_i - (\theta_0 + \theta_1 x_i)|}{\Delta_i} \right). \tag{8.8}$$

For the case of Gaussian homoscedastic uncertainties, the minimization of eq. 8.7 simplifies to

$$\theta_1 = \frac{\sum_i^N x_i y_i - \bar{x}\bar{y}}{\sum_i^N (x_i - \bar{x})^2}, \tag{8.9}$$

$$\theta_0 = \bar{y} - \theta_1 \bar{x}, \tag{8.10}$$

where $\bar{x}$ is the mean value of x and $\bar{y}$ is the mean value of y. As an illustration, these estimates of θ_0 and θ_1 correspond to the center of the ellipse shown in the bottom-left panel in figure 8.1. An estimate of the variance associated with this regression and the standard errors on the estimated parameters are given by

$$\sigma^2 = \sum_{i=1}^{N} (y_i - \theta_0 + \theta_1 x_i)^2, \tag{8.11}$$

$$\sigma_{\theta_1}^2 = \sigma^2 \frac{1}{\sum_i^N (x_i - \bar{x})^2}, \tag{8.12}$$

$$\sigma_{\theta_0}^2 = \sigma^2 \left(\frac{1}{N} + \frac{\bar{x}^2}{\sum_i^N (x_i - \bar{x})^2} \right). \tag{8.13}$$

For heteroscedastic errors, and in general for more complex regression functions, it is easier and more compact to generalize regression in terms of matrix notation. We, therefore, define regression in terms of a design matrix, M, such that

$$Y = M\boldsymbol{\theta}, \tag{8.14}$$

where Y is an N-dimensional vector of values y_i,

$$Y = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ . \\ y_{N-1} \end{bmatrix}. \tag{8.15}$$

For our straight-line regression function, $\boldsymbol{\theta}$ is a two-dimensional vector of regression coefficients,

$$\boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix}, \tag{8.16}$$

and M is a $2 \times N$ matrix,

$$M = \begin{bmatrix} 1 & x_0 \\ 1 & x_1 \\ 1 & x_2 \\ . & . \\ 1 & x_{N-1} \end{bmatrix}, \tag{8.17}$$

where the constant value in the first column captures the θ_0 term in the regression.

For the case of heteroscedastic uncertainties, we define a covariance matrix, C, as an $N \times N$ matrix,

$$C = \begin{bmatrix} \sigma_0^2 & 0 & . & 0 \\ 0 & \sigma_1^2 & . & 0 \\ . & . & . & . \\ 0 & 0 & . & \sigma_{N-1}^2 \end{bmatrix} \tag{8.18}$$

with the diagonals of this matrix containing the uncertainties, σ_i, on the dependent variable, Y.

The maximum likelihood solution for this regression is

$$\boldsymbol{\theta} = (M^T C^{-1} M)^{-1} (M^T C^{-1} Y), \tag{8.19}$$

which again minimizes the sum of the square errors, $(Y - \boldsymbol{\theta} M)^T C^{-1} (Y - \boldsymbol{\theta} M)$, as we did explicitly in eq. 8.9. The uncertainties on the regression coefficients, $\boldsymbol{\theta}$, can now be expressed as the symmetric matrix

$$\Sigma_\theta = \begin{bmatrix} \sigma_{\theta_0}^2 & \sigma_{\theta_0\theta_1} \\ \sigma_{\theta_0\theta_1} & \sigma_{\theta_1}^2 \end{bmatrix} = [M^T C^{-1} M]^{-1}. \tag{8.20}$$

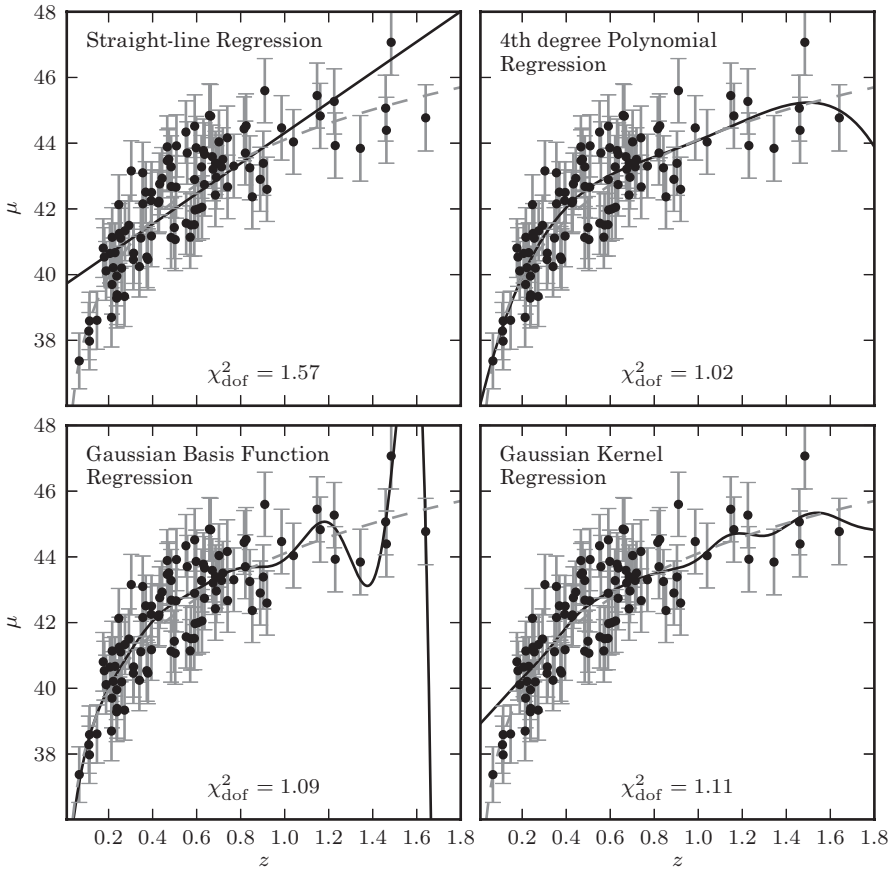


Figure 8.2. Various regression fits to the distance modulus vs. redshift relation for a simulated set of 100 supernovas, selected from a distribution $p(z) \propto (z/z_0)^2 \exp[(z/z_0)^{1.5}]$ with $z_0 = 0.3$. Gaussian basis functions have 15 Gaussians evenly spaced between $z = 0$ and 2, with widths of 0.14. Kernel regression uses a Gaussian kernel with width 0.1.

Whether we have sufficient data to constrain the regression (i.e., sufficient degrees of freedom) is defined by whether $M^T M$ is an invertible matrix.

The top-left panel of figure 8.2 illustrates a simple linear regression of redshift, z, against distance modulus, μ, for the set of 100 supernovas described in § 8.1.1. The solid line shows the regression function for the straight-line model and the dashed line the underlying cosmological model from which the data were drawn (which of course cannot be described by a straight line). It is immediately apparent that the chosen regression model does not capture the structure within the data at the high and low redshift limits—the model does not have sufficient flexibility to reproduce the correlation displayed by the data. This is reflected in the $\chi^2_{\rm dof}$ for this fit which is 1.54 (see § 4.3.1 for a discussion of the interpretation of $\chi^2_{\rm dof}$).

We now relax the assumptions we made at the start of this section, allowing not just for heteroscedastic uncertainties but also for correlations between the measures of the dependent variables. With no loss in generality, eq. 8.19 can be extended to allow for covariant data through the off-diagonal elements of the covariance matrix C.

8.2.1. Multivariate Regression

For multivariate data (where we fit a hyperplane rather than a straight line) we simply extend the description of the regression function to multiple dimensions, with $y = f(x|\boldsymbol{\theta})$ given by

$$y_i = \theta_0 + \theta_1 x_{i1} + \theta_2 x_{i2} + \cdots + \theta_k x_{ik} + \epsilon_i \tag{8.21}$$

with θ_i the regression parameters and x_{ik} the kth component of the ith data entry within a multivariate data set. This multivariate regression follows naturally from the definition of the design matrix with

$$M = \begin{pmatrix} 1 & x_{01} & x_{02} & . & x_{0k} \\ 1 & x_{11} & x_{12} & . & x_{1k} \\ . & . & . & . & . \\ 1 & x_{N1} & x_{N2} & . & x_{Nk} \end{pmatrix}. \tag{8.22}$$

The regression coefficients (which are estimates of $\boldsymbol{\theta}$ and are often differentiated from the true values by writing them as $\hat{\boldsymbol{\theta}}$) and their uncertainties are, as before,

$$\boldsymbol{\theta} = (M^T C^{-1} M)^{-1} (M^T C^{-1} Y) \tag{8.23}$$

and

$$\Sigma_{\boldsymbol{\theta}} = [M^T C^{-1} M]^{-1}. \tag{8.24}$$

Multivariate linear regression with homoscedastic errors on dependent variables can be performed using the routine `sklearn.linear_model.LinearRegression`. For data with homoscedastic errors, AstroML implements a similar routine:

```
import numpy as np
from astroML.linear_model import LinearRegression
X = np.random.random((100, 2))  # 100 points in
    2 dimensions
dy = np.random.random(100)  # heteroscedastic errors
y = np.random.normal(X[:, 0] + X[:, 1], dy)

model = LinearRegression()
model.fit(X, y, dy)
y_pred = model.predict(X)
```

`LinearRegression` in Scikit-learn has a similar interface, but does not explicitly account for heteroscedastic errors. For a more realistic example, see the source code of figure 8.2.

8.2.2. Polynomial and Basis Function Regression

Due to its simplicity, the derivation of regression in most textbooks is undertaken using a straight-line fit to the data. However, the straight line can simply be interpreted as a first-order expansion of the regression function $y = f(x|\boldsymbol{\theta})$. In general we can express $f(x|\boldsymbol{\theta})$ as the sum of arbitrary (often nonlinear) functions as long as the model is linear in terms of the regression parameters, $\boldsymbol{\theta}$. Examples of these general linear models include a Taylor expansion of $f(x)$ as a series of polynomials where we solve for the amplitudes of the polynomials, or a linear sum of Gaussians with fixed positions and variances where we fit for the amplitudes of the Gaussians.

Let us initially consider polynomial regression and write $f(x|\boldsymbol{\theta})$ as

$$y_i = \theta_0 + \theta_1 x_i + \theta_2 x_i^2 + \theta_3 x_i^3 + \cdots . \tag{8.25}$$

The design matrix for this expansion becomes

$$M = \begin{pmatrix} 1 & x_0 & x_0^2 & x_0^3 \\ 1 & x_1 & x_1^2 & x_1^3 \\ . & . & . & . \\ 1 & x_N & x_N^2 & x_N^3 \end{pmatrix}, \tag{8.26}$$

where the terms in the design matrix are 1, x, x^2, and x^3, respectively. The solution for the regression coefficients and the associated uncertainties are again given by eqs. 8.19 and 8.20.

A fourth-degree polynomial fit to the supernova data is shown in the top-right panel of figure 8.2. The increase in flexibility of the model improves the fit (note that we have to be aware of overfitting the data if we just arbitrarily increase the degree of the polynomial; see § 8.11). The $\chi^2_{\rm dof}$ of the regression is 1.02, which indicates a much better fit than the straight-line case. At high redshift, however, there is a systematic deviation between the polynomial regression and the underlying generative model (shown by the dashed line), which illustrates the danger of extrapolating this model beyond the range probed by the data.

Polynomial regression with heteroscedastic errors can be performed using the `PolynomialRegression` function in AstroML:

```
import numpy as np
from astroML.linear_model import PolynomialRegression

X = np.random.random((100, 2)) # 100 points in 2 dims
y = X[:, 0] ** 2 + X[:, 1] ** 3
model = PolynomialRegression(3)
        # fit 3rd degree polynomial
model.fit(X, y)
y_pred = model.predict(X)
```

Here we have used homoscedastic errors for simplicity. Heteroscedastic errors in `y` can be used in a similar way to `LinearRegression`, above. For a more realistic example, see the source code of figure 8.2.

The number of terms in the polynomial regression grows exponentially with order. Given a data set with k dimensions to which we fit a p-dimensional polynomial, the number of parameters in the model we are fitting is given by

$$m = \frac{(p+k)!}{p!\,k!}, \tag{8.27}$$

including the intercept or offset. The number of degrees of freedom for the regression model is then $\nu = N - m$ and the probability of that model is given by a χ^2 distribution with ν degrees of freedom.

We can generalize the polynomial model to a basis function representation by noting that each row of the design matrix can be replaced with any series of linear or nonlinear functions of the variables x_i. Despite the use of arbitrary basis functions, the resulting problem remains linear, because we are fitting only the coefficients multiplying these terms. Examples of commonly used basis functions include Gaussians, trigonometric functions, inverse quadratic functions, and splines.

Basis function regression can be performed using the routine `BasisFunctionRegression` in AstroML. For example, Gaussian basis function regression is as follows:

```
import numpy as np
from astroML.linear_model import
     BasisFunctionRegression

X = np.random.random((100, 1))  # 100 points in 1
    # dimension dy = 0.1
y = np.random.normal(X[:, 0], dy)
mu = np.linspace(0, 1, 10)[:, np.newaxis]
     # 10 x 1 array of mu
sigma = 0.1

model = BasisFunctionRegression('gaussian', mu=mu,
        sigma=sigma)
model.fit(X, y, dy)
y_pred = model.predict(X)
```

For a further example, see the source code of figure 8.2.

The application of Gaussian basis functions to our example regression problem is shown in figure 8.2. In the lower-left panel, 15 Gaussians, evenly spaced between redshifts $0 < z < 2$ with widths of $\sigma_z = 0.14$, are fit to the supernova data. The $\chi^2_{\rm dof}$ for this fit is 1.09, comparable to that for polynomial regression.

8.3. Regularization and Penalizing the Likelihood

All regression examples so far have sought to minimize the mean square errors between a model and data with known uncertainties. The Gauss–Markov theorem states that this least-squares approach results in the minimum variance unbiased estimator (see § 3.2.2) for the linear model. In some cases, however, the regression problem may be ill posed and the best unbiased estimator is not the most appropriate regression. Instead, we can trade an increase in bias for a reduction in variance. Examples of such cases include data that are highly correlated (which results in ill-conditioned matrices), or when the number of terms in the regression model decreases the number of degrees of freedom such that we must worry about overfitting of the data.

One solution to these problems is to penalize or limit the complexity of the underlying regression model. This is often referred to as regularization, or shrinkage, and works by applying a penalty to the likelihood function. Regularization can come in many forms, but usually imposes smoothness on the model, or limits the numbers of, or the values of, the regression coefficients.

In § 8.2 we showed that regression minimizes the least-squares equation,

$$(Y - M\boldsymbol{\theta})^T(Y - M\boldsymbol{\theta}). \tag{8.28}$$

We can impose a penalty on this minimization if we include a regularization term,

$$(Y - M\boldsymbol{\theta})^T(Y - M\boldsymbol{\theta}) + \lambda|\boldsymbol{\theta}^T\boldsymbol{\theta}|, \tag{8.29}$$

where λ is the regularization or smoothing parameter and $|\boldsymbol{\theta}^T\boldsymbol{\theta}|$ is an example of the penalty function. In this example, we penalize the size of the regression coefficients (which is known as ridge regression as we will discuss in the next section). Solving for $\boldsymbol{\theta}$ we arrive at a modification of eq. 8.19,

$$\boldsymbol{\theta} = (M^TC^{-1}M + \lambda I)^{-1}(M^TC^{-1}Y), \tag{8.30}$$

where I is the identity matrix. One aspect worth noting about robustness through regularization is that, even if $M^TC^{-1}M$ is singular, solutions can still exist for $(M^TC^{-1}M + \lambda I)$.

A Bayesian implementation of regularization would use the prior to impose constraints on the probability distribution of the regression coefficients. If, for example, we assumed that the prior on the regression coefficients was Gaussian with the width of this Gaussian governed by the regularization parameter λ then we could write it as

$$p(\boldsymbol{\theta}|I) \propto \exp\left(\frac{-(\lambda\boldsymbol{\theta}^T\boldsymbol{\theta})}{2}\right). \tag{8.31}$$

Multiplying the likelihood function by this prior results in a posterior distribution with an exponent $(Y - M\boldsymbol{\theta})^T(Y - M\boldsymbol{\theta}) + \lambda|\boldsymbol{\theta}^T\boldsymbol{\theta}|$, equivalent to the MLE regularized regression described above. This Gaussian prior corresponds to ridge regression. For LASSO regression, described below, the corresponding prior would be an exponential (Laplace) distribution.

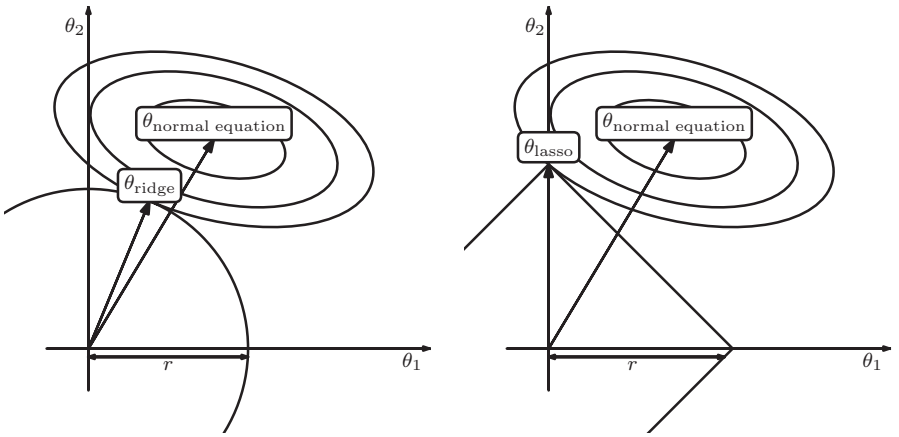


Figure 8.3. A geometric interpretation of regularization. The right panel shows L_1 regularization (LASSO regression) and the left panel L_2 regularization (ridge regularization). The ellipses indicate the posterior distribution for no prior or regularization. The solid lines show the constraints due to regularization (limiting $\boldsymbol{\theta}^2$ for ridge regression and $|\boldsymbol{\theta}|$ for LASSO regression). The corners of the L_1 regularization create more opportunities for the solution to have zeros for some of the weights.

8.3.1. Ridge Regression

The regularization example above is often referred to as ridge regression or Tikhonov regularization [22]. It provides a penalty on the sum of the squares of the regression coefficients such that

$$|\boldsymbol{\theta}|^2 < s, \tag{8.32}$$

where s controls the complexity of the model in the same way as the regularization parameter λ in eq. 8.29. By suppressing large regression coefficients this penalty limits the variance of the system at the expense of an increase in the bias of the derived coefficients.

A geometric interpretation of ridge regression is shown in figure 8.3. The solid elliptical contours are the likelihood surface for the regression with no regularization. The circle illustrates the constraint on the regression coefficients ($|\boldsymbol{\theta}|^2 < s$) imposed by the regularization. The penalty on the likelihood function, based on the squared norm of the regression coefficients, drives the solution to small values of $\boldsymbol{\theta}$. The smaller the value of s (or the larger the regularization parameter λ) the more the regression coefficients are driven toward zero.

The regularized regression coefficients can be derived through matrix inversion as before. Applying an SVD to the $N \times m$ design matrix (where m is the number of terms in the model; see § 8.2.2) we get $M = U\Sigma V^T$, with U an $N \times m$ matrix, V^T the $m \times m$ matrix of eigenvectors and Σ the $m \times m$ matrix of eigenvalues. We can now write the regularized regression coefficients as

$$\boldsymbol{\theta} = V\Sigma' U^T Y, \tag{8.33}$$

where Σ' is a diagonal matrix with elements $d_i/(d_i^2 + \lambda)$, with d_i the eigenvalues of MM^T.

As λ increases, the diagonal components are down weighted so that only those components with the highest eigenvalues will contribute to the regression. This relates directly to the PCA analysis we described in § 7.3. Projecting the variables onto the eigenvectors of MM^T such that

$$Z = MV, \tag{8.34}$$

with z_i the ith eigenvector of M, ridge regression shrinks the regression coefficients for any component for which its eigenvalues (and therefore the associated variance) are small.

The effective goodness of fit for a ridge regression can be derived from the response of the regression function,

$$\hat{y} = M(M^T M + \lambda I)^{-1} M^T y, \tag{8.35}$$

and the number of degrees of freedom,

$$\text{DOF} = \text{Trace}[M(M^T M + \lambda I)^{-1} M^T] = \sum_i \frac{d_i^2}{d_i^2 + \lambda}. \tag{8.36}$$

Ridge regression can be accomplished with the `Ridge` class in Scikit-learn:

```
import numpy as np
from sklearn.linear_model import Ridge

X = np.random.random((100, 10))
    # 100 points in 10 dims
y = np.dot(X, np.random.random(10))
    # random combination of X
model = Ridge(alpha = 0.05)  # alpha controls
        # regularization
model.fit(X, y)
y_pred = model.predict(X)
```

For more information, see the Scikit-learn documentation.

Figure 8.4 uses the Gaussian basis function regression of § 8.2.2 to illustrate how ridge regression will constrain the regression coefficients. The left panel shows the general linear regression for the supernovas (using 100 evenly spaced Gaussians with $\sigma = 0.2$). As we noted in § 8.2.2, an increase in the number of model parameters results in an overfitting of the data (the lower panel in figure 8.4 shows how the regression coefficients for this fit are on the order of 10^8). The central panel demonstrates how ridge regression (with $\lambda = 0.005$) suppresses the amplitudes of the regression coefficients and the resulting fluctuations in the modeled response.

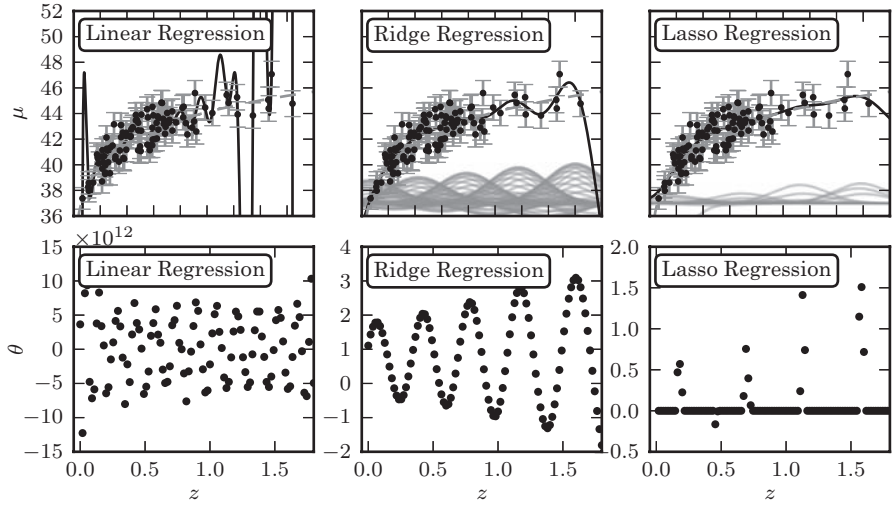


Figure 8.4. Regularized regression for the same sample as Fig 8.2. Here we use Gaussian basis function regression with a Gaussian of width $\sigma = 0.2$ centered at 100 regular intervals between $0 \leq z \leq 2$. The lower panels show the best-fit weights as a function of basis function position. The left column shows the results with no regularization: the basis function weights w are on the order of 10^8, and overfitting is evident. The middle column shows ridge regression (L_2 regularization) with $\lambda = 0.005$, and the right column shows LASSO regression (L_1 regularization) with $\lambda = 0.005$. All three methods are fit without the bias term (intercept).

8.3.2. LASSO Regression

Ridge regression uses the square of the regression coefficients to regularize the fits (i.e., the L_2 norm). A modification of this approach is to use the L_1 norm [2] to subset the variables within a model as well as applying shrinkage. This technique is known as LASSO (least absolute shrinkage and selection; see [21]). LASSO penalizes the likelihood as

$$(Y - M\boldsymbol{\theta})^T(Y - M\boldsymbol{\theta}) + \lambda|\boldsymbol{\theta}|, \tag{8.37}$$

where $|\boldsymbol{\theta}|$ penalizes the absolute value of $\boldsymbol{\theta}$. LASSO regularization is equivalent to least-squares regression with a penalty on the absolute value of the regression coefficients,

$$|\boldsymbol{\theta}| < s. \tag{8.38}$$

The most interesting aspect of LASSO is that it not only weights the regression coefficients, it also imposes sparsity on the regression model. Figure 8.3 illustrates the impact of the L_1 norm on the regression coefficients from a geometric perspective. The $\lambda|\boldsymbol{\theta}|$ penalty preferentially selects regions of likelihood space that coincide with one of the vertices within the region defined by the regularization. This corresponds to setting one (or more if we are working in higher dimensions) of the model attributes to zero. This subsetting of the model attributes reduces the underlying complexity of the model (i.e., we make zeroing of weights, or feature selection, more

aggressive). As λ increases, the size of the region encompassed within the constraint decreases.

Ridge regression can be accomplished with the `Lasso` class in Scikit-learn:

```
import numpy as np
from sklearn.linear_model import Lasso

X = np.random.random((100, 10))
    # 100 points in 10 dims
y = np.dot(X, np.random.random(10))
    # random comb. of X
model = Lasso(alpha = 0.05)  # alpha controls
        # regularization
model.fit(X, y)
y_pred = model.predict(X)
```

For more information, see the Scikit-learn documentation.

Figure 8.4 shows this effect for the supernova data. Of the 100 Gaussians in the input model, with $\lambda = 0.005$, only 13 are selected by LASSO (note the regression coefficients in the lower panel). This reduction in model complexity suppresses the overfitting of the data.

A disadvantage of LASSO is that, unlike ridge regression, there is no closed-form solution. The optimization becomes a quadratic programming problem (though it is still a convex optimization). There are a number of numerical techniques that have been developed to address these issues including coordinate-gradient descent [12] and least angle regression [5].

8.3.3. How Do We Choose the Regularization Parameter λ?

In each of the regularization examples above we defined a "shrinkage parameter" that we refer to as the regularization parameter. The natural question then is how do we set λ? So far we have only noted that as we increase λ we increase the constraints on the range regression coefficients (with $\lambda = 0$ returning the standard least-squares regression). We can, however, evaluate its impact on the regression as a function of its amplitude.

Applying the k-fold cross-validation techniques described in § 8.11 we can define an error (for a specified value of λ) as

$$\text{Error}(\lambda) = k^{-1} \sum_k N_k^{-1} \sum_i^{N_k} \frac{[y_i - f(x_i|\boldsymbol{\theta})]^2}{\sigma_i^2}, \tag{8.39}$$

where N_k^{-1} is the number of data points in the kth cross-validation sample, and the summation over N_k represents the sum of the squares of the residuals of the fit. Estimating λ is then simply a case of finding the λ that minimizes the cross-validation error.

8.4. Principal Component Regression

For the case of high-dimensional data or data sets where the variables are collinear, the relation between ridge regression and principal component analysis can be exploited to define a regression based on the principal components. For centered data (i.e., zero mean) we recall from § 7.3 that we can define the principal components of a system from the data covariance matrix, $X^T X$, by applying an eigenvalue decomposition (EVD) or singular value decomposition (SVD),

$$X^T X = V \Sigma V^T, \tag{8.40}$$

with V^T the eigenvectors and Σ the eigenvalues.

Projecting the data matrix onto these eigenvectors we define a set of projected data points,

$$Z = X V^T, \tag{8.41}$$

and truncate this expansion to exclude those components with small eigenvalues. A standard linear regression can now be applied to the data transposed to this principal component space with

$$Y = M_z \boldsymbol{\theta} + \epsilon, \tag{8.42}$$

with M_z the design matrix for the projected components z_i. The PCA analysis (including truncation) and the regression are undertaken as separate steps in this procedure. The distinction between principal component regression (PCR) and ridge regression is that the number of model components in PCR is ordered by their eigenvalues and is absolute (i.e., we weight the regression coefficients by 1 or 0). For ridge regression the weighting of the regression coefficients is continuous.

The advantages of PCR over ridge regression arise primarily for data containing independent variables that are collinear (i.e., where the correlation between the variables is almost unity). For these cases, the regression coefficients have large variance and their solutions can become unstable. Excluding those principal components with small eigenvalues alleviates this issue. At what level to truncate the set of eigenvectors is, however, an open question (see § 7.3.2). A simple approach to take is to truncate based on the eigenvalue with common cutoffs ranging between 1% and 10% of the average eigenvalue (see § 8.2 of [10] for a detailed discussion of truncation levels for PCR). The disadvantage of such an approach is that an eigenvalue does not always correlate with the ability of a given principal component to predict the dependent variable. Other techniques, including cross-validation [17], have been proposed yet there is no well-adopted solution to this problem.

Finally, we note that in the case of ill-conditioned regression problems (e.g., those with collinear data), most implementations of linear regression will implicitly perform a form of principal component regression when inverting the singular matrix $M^T M$. This comes through the use of the robust pseudoinverse, which truncates small singular values to prevent numerical overflow in ill-conditioned problems.

8.5. Kernel Regression

The previous sections found the regression or objective function that "best fits" a set of data assuming a linear model. Before we address the question of nonlinear optimization we will describe a number of techniques that make use of locality within the data (i.e., local regression techniques).

Kernel or Nadaraya–Watson [18, 23] regression defines a kernel, $K(x_i, x)$, local to each data point, with the amplitude of the kernel depending only on the distance from the local point to all other points in the sample. The properties of the kernel are such that it is positive for all values and asymptotes to zero as the distance approaches infinity. The influence of the kernel (i.e., the region of parameter space over which we weight the data) is determined by its width or bandwidth, h. Common forms of the kernel include the top-hat function, and the Gaussian distribution.

The Nadaraya–Watson estimate of the regression function is given by

$$f(x|K) = \frac{\sum_{i=1}^{N} K\left(\frac{||x_i - x||}{h}\right) y_i}{\sum_{i=1}^{N} K\left(\frac{||x_i - x||}{h}\right)}, \tag{8.43}$$

which can be viewed as taking a weighted average of the dependent variable, y. This gives higher weight to points near x with a weighting function,

$$w_i(x) = \frac{K\left(\frac{||x_i - x||}{h}\right)}{\sum_{i=1}^{N} K\left(\frac{||x_i - x||}{h}\right)}. \tag{8.44}$$

Nadaraya–Watson kernel regression can be performed using AstroML in the following way:

```
import numpy as np
from astroML.linear_model import NadarayaWatson

X = np.random.random((100, 2))# 100 points in 2 dims
y = X[:, 0] + X[:, 1]
model = NadarayaWatson('gaussian', 0.05)
model.fit(X, y)
y_pred = model.predict(X)
```

Figure 8.2 shows the application of Gaussian kernel regression to the synthetic supernova data compared to the standard linear regression techniques introduced in § 8.2. For this example we use a Gaussian kernel with $h = 0.1$ that is constant across the redshift interval. At high redshift, where the data provide limited support for the model, we see that the weight function drives the predicted value of y to that of the nearest neighbor. This prevents the extrapolation problems common when fitting polynomials that are not constrained at the edges of the data (as we saw in the top panels of figure 8.2). The width of the kernel acts as a smoothing function.

At low redshift, the increase in the density of points at $z \approx 0.25$ biases the weighted estimate of $\hat{y}$ for $z < 0.25$. This is because, while the kernel amplitude is small for the higher redshift points, the increase in the sample size results in a higher than average weighting of points at $z \approx 0.25$. Varying the bandwidth as a function of the independent variable can correct for this. The rule of thumb for kernel-based regression (as with kernel density estimation described in § 6.3) is that the bandwidth is more important that the exact shape of the kernel.

Estimation of the optimal bandwidth of the kernel is straightforward using cross-validation (see § 8.11). We define the CV error as

$$\mathrm{CV}_{L_2}(h) = \frac{1}{N}\sum_{i=1}^{N}\left(y_i - f\left(x_i|K\left(\frac{||x_i - x_j||}{h}\right)\right)\right)^2. \tag{8.45}$$

We actually do not need to compute separate estimates for each leave-one-out cross-validation subsample if we rewrite this as

$$\mathrm{CV}_{L_2}(h) = \frac{1}{N}\frac{\sum_{i=1}^{N}(y_i - f(x_i|K))^2}{\left(1 - \frac{K(0)}{\sum_{j=1}^{N}K\left(\frac{||x_i - x_j||}{h}\right)}\right)^2}. \tag{8.46}$$

It can be shown that, as for kernel density estimation, the optimal bandwidth decreases with sample size at a rate of $N^{-1/5}$.

8.6. Locally Linear Regression

Related to kernel regression is *locally linear regression*, where we solve a separate weighted least-squares problem at each point x, finding the $w(x)$ which minimizes

$$\sum_{i=1}^{N} K\left(\frac{||x - x_i||}{h}\right)(y_i - w(x)\,x_i)^2. \tag{8.47}$$

The assumption for locally linear regression is that the regression function can be approximated by a Taylor series expansion about any local point. If we truncated this expansion at the first term (i.e., a locally constant solution) we recover kernel regression. For locally linear regression the function estimate is

$$\begin{aligned} f(x|K) &= \theta(x)x & (8.48)\\ &= x^T\left(\mathbf{X}^T W(x)\mathbf{X}\right)^{-1}\mathbf{X}^T W(x)\mathbf{Y} & (8.49)\\ &= \sum_{i=1}^{N} w_i(x)y_i, & (8.50) \end{aligned}$$

where $W(x)$ is an $N \times N$ diagonal matrix with the ith diagonal element given by $K||x_i - x||/h$.

A common form for $K(x)$ is the tricubic kernel,

$$K(x_i, x) = \left(1 - \left(\frac{|x - x_i|}{h}\right)^3\right)^3 \tag{8.51}$$

for $|x_i - x| < h$, which is often referred to as lowess (or loess; locally weighted scatter plot smoothing); see [3].

There are further extensions possible for local linear regression:

- *Local polynomial regression*. We can consider any polynomial order. However there is a bias–variance (complexity) trade-off, as usual. The general consensus is that going past linear increases variance without decreasing bias much, since local linear regression captures most of the boundary bias.
- *Variable-bandwidth kernels*. Let the bandwidth for each training point be inversely proportional to its kth nearest neighbor's distance. Generally a good idea in practice, though there is less theoretical consensus on how to choose the parameters in this framework.

None of these modifications improves the convergence rate.

8.7. Nonlinear Regression

Forcing data to correspond to a linear model through the use of coordinate transformations is a well-used trick in astronomy (e.g., the extensive use of logarithms in the astronomical literature to linearize complex relations between attributes; fitting $y = A\exp(Bx)$ becomes a linear problem with $z = K + Bx$, where $z = \log y$ and $K = \log A$). These simplifications, while often effective, introduce other complications (including the non-Gaussian nature of the uncertainties for low signal-to-noise data). We must, eventually, consider the case of nonlinear regression and model fitting.

In the cosmological examples described previously we have fit a series of parametric and nonparametric models to the supernova data. Given that we know the theoretical form of the underlying cosmological model, these models are somewhat ad hoc (e.g., using a series of polynomials to parameterize the dependence of distance modulus on cosmological redshift). In the following we consider directly fitting the cosmological model described in eq. 8.4. Solving for Ω_m and Ω_Λ is a nonlinear optimization problem requiring that we maximize the posterior,

$$p(\Omega_m, \Omega_\Lambda|z, I) \propto \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}\sigma_i} \exp\left(\frac{-(\mu_i - \mu(z_i|\Omega_m, \Omega_\Lambda))^2}{2\sigma_i^2}\right) p(\Omega_m, \Omega_\Lambda) \tag{8.52}$$

with μ_i the distance modulus for the supernova and z_i the redshift.

In § 5.8 we introduced Markov chain Monte Carlo as a sampling technique that can be used for searching through parameter space. Figure 8.5 shows the resulting likelihood contours for our cosmological model after applying the Metropolis–Hastings algorithm to generate the MCMC chains and integrating the chains over the parameter space.

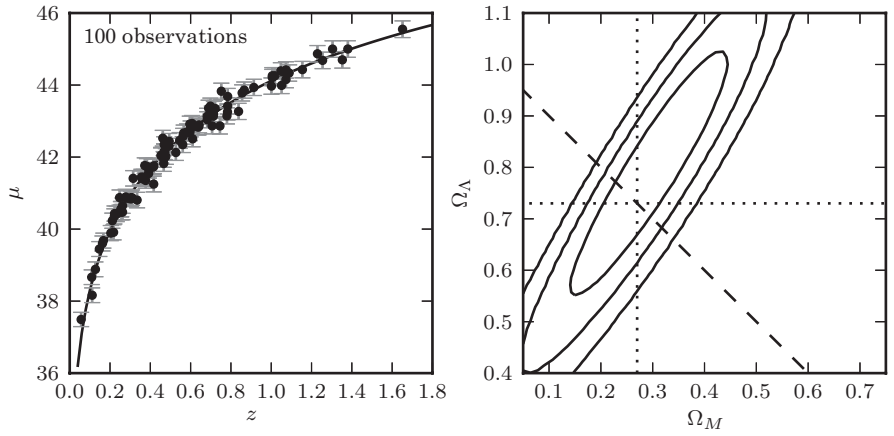


Figure 8.5. Cosmology fit to the standard cosmological integral. Errors in μ are a factor of ten smaller than for the sample used in figure 8.2. Contours are 1σ, 2σ, and 3σ for the posterior (uniform prior in Ω_M and Ω_Λ). The dashed line shows flat cosmology. The dotted lines show the input values.

An alternate approach is to use the Levenberg–Marquardt (LM) algorithm [15, 16] to optimize the maximum likelihood estimation. LM searches for the sum-of-squares minima of a multivariate distribution through a combination of gradient descent and Gauss–Newton optimization. Assuming that we can express our regression function as a Taylor series expansion then, to first order, we can write

$$f(x_i|\boldsymbol{\theta}) = f(x_i|\boldsymbol{\theta}_0) + J\, d\boldsymbol{\theta}, \tag{8.53}$$

where $\boldsymbol{\theta}_0$ is an initial guess for the regression parameters, J is the Jacobian about this point ($J = \partial f(x_i|\boldsymbol{\theta})/\partial\boldsymbol{\theta}$), and $d\boldsymbol{\theta}$ is a small change in the regression parameters. LM minimizes the sum of square errors,

$$\sum_i (y_i - f(x_i|\boldsymbol{\theta}_0) - J_i d\boldsymbol{\theta})^2 \tag{8.54}$$

for the perturbation $d\boldsymbol{\theta}$. This results in an update relation for $d\boldsymbol{\theta}$ of

$$(J^T C^{-1} J + \lambda\, \mathrm{diag}(J^T C^{-1} J))\, d\boldsymbol{\theta} = J^T C^{-1}(Y - f(X|\boldsymbol{\theta})), \tag{8.55}$$

with C the standard covariance matrix introduced in eq. 8.18. In this expression the λ term acts as a damping parameter (in a manner similar to ridge regression regularization discussed in § 8.3.1). For small λ the relation approximates a Gauss–Newton method (i.e., it minimizes the parameters assuming the function is quadratic). For large λ the perturbation $d\boldsymbol{\theta}$ follows the direction of steepest descent. The $\mathrm{diag}(J^T C^{-1} J)$ term, as opposed to the identity matrix used in ridge regression, ensures that the update of $d\boldsymbol{\theta}$ is largest along directions where the gradient is smallest (which improves convergence).

LM is an iterative process. At each iteration LM searches for the step $d\boldsymbol{\theta}$ that minimizes eq. 8.54 and then updates the regression model. The iterations cease when

the step size, or change in likelihood values reaches a predefined value. Throughout the iteration process the damping parameter, λ, is adaptively varied (decreasing as the minimum is approached). A common approach involves decreasing (or increasing) λ by ν^k, where k is the number of the iteration and ν is a damping factor (with $\nu > 1$). Upon successful convergence the regression parameter covariances are given by $[J^T \boldsymbol{\theta} J]^{-1}$. It should be noted that the success of the LM algorithm often relies on the initial guesses for the regression parameters being close to the maximum likelihood solution (in the presence of nonlocal minima), or the likelihood function surface being unimodal.

The submodule `scipy.optimize` includes several routines for optimizing linear and nonlinear equations. The Levenberg–Marquardt algorithm is used in `scipy.optimize.leastsq`. Below is a brief example of using the routine to estimate the first six terms of the Taylor series for the function $y = \sin x \approx \sum_n a_n x^n$:

```
import numpy as np
from scipy import optimize
x = np.linspace(-3, 3, 100) # 100 values between -3
                            # and 3
def taylor_err(a, x, f):
    p = np.arange(len(a))[:, np.newaxis]
        # column vector
    return f(x) - np.dot(a, x ** p)
a_start = np.zeros(6)  # starting guess
a_best, flag = optimize.leastsq(taylor_err, a_start,
                                args=(x, np.sin))
```

8.8. Uncertainties in the Data

In the opening section we introduced the problem of regression in its most general form. Computational complexity (particularly for the case of multivariate data) led us through a series of approximations that can be used in optimizing the likelihood and the prior (e.g., that the uncertainties have a Gaussian distribution, that the independent variables are error-free, that we can control complexity of the model, and that we can express the likelihood in terms of linear functions). Eventually we have to face the problem that many of these assumptions no longer hold for data in the wild. We have addressed the question of model complexity; working from linear to nonlinear regression. We now return to the question of *error behavior* and consider the uncertainty that is inherent in any analysis.

8.8.1. Uncertainties in the Dependent and Independent Axes

In almost all real-world applications, the assumption that one variable (the independent variable) is essentially free from any uncertainty is not valid. Both the dependent and independent variables will have measurement uncertainties. For our example data set we have assumed that the redshifts of the supernovas are known to a very high level of accuracy (i.e., that we measure these redshifts spectroscopically). If, for example, the redshifts of the supernovas were estimated based on the colors of the supernova (or host galaxy) then the errors on the redshift estimate can be significant. For our synthetic data we assume fractional uncertainties on the independent variable of 10%.

The impact of errors on the "independent" variables is a bias in the derived regression coefficients. This is straightforward to show if we consider a linear model with a dependent and independent variable, y^* and x^*. We can write the objective function as before,

$$y_i^* = \theta_0 + \theta_1 x_i^*. \tag{8.56}$$

Now let us assume that we observe y and x, which are noisy representations of y^* and x^*, i.e.,

$$x_i = x_i^* + \delta_i, \tag{8.57}$$
$$y_i = y^* + \epsilon_i, \tag{8.58}$$

with δ and ϵ centered normal distributions.

Solving for y we get

$$y = \theta_0 + \theta_1(x_i - \delta_i) + \epsilon_i. \tag{8.59}$$

The uncertainty in x is now part of the regression equation and scales with the regression coefficients (biasing the regression coefficient). This problem is known in the statistics literature as *total least squares* and belongs to the class of "errors-in-variables" problems; see [4]. A very detailed discussion of regression and the problem of uncertainties from an astronomical perspective can be found in [8, 11].

How can we account for the measurement uncertainties in both the independent and dependent variables? Let us start with a simple example where we assume the errors are Gaussian so we can write the covariance matrix as

$$\Sigma_i = \begin{bmatrix} \sigma_{x_i}^2 & \sigma_{xy_i} \\ \sigma_{xy_i} & \sigma_{y_i}^2 \end{bmatrix}. \tag{8.60}$$

For a straight-line regression we express the slope of the line, θ_1, in terms of its normal vector,

$$\mathbf{n} = \begin{bmatrix} -\sin\alpha \\ \cos\alpha \end{bmatrix}, \tag{8.61}$$

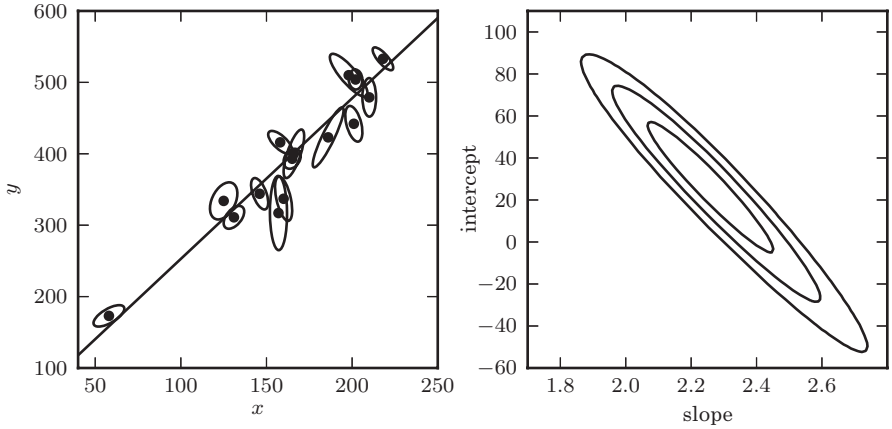


Figure 8.6. A linear fit to data with correlated errors in x and y. In the literature, this is often referred to as *total least squares* or *errors-in-variables* fitting. The left panel shows the lines of best fit; the right panel shows the likelihood contours in slope/intercept space. The points are the same set used for the examples in [8].

where $\theta_1 = \arctan(\alpha)$ and α is the angle between the line and the x-axis. The covariance matrix projects onto this space as

$$S_i^2 = \mathbf{n}^T \Sigma_i \mathbf{n} \tag{8.62}$$

and the distance between a point and the line is given by (see [8])

$$\Delta_i = \mathbf{n}^T z_i - \theta_0 \cos\alpha, \tag{8.63}$$

where z_i represents the data point (x_i, y_i). The log-likelihood is then

$$\ln \mathrm{L} = -\sum_i \frac{\Delta_i^2}{2S_i^2}. \tag{8.64}$$

Maximizing this likelihood for the regression parameters, θ_0 and θ_1 is shown in figure 8.6, where we use the data from [8] with correlated uncertainties on the x and y components, and recover the underlying linear relation. For a single parameter search (θ_1) the regression can be undertaken in a brute-force manner. As we increase the complexity of the model or the dimensionality of the data, the computational cost will grow and techniques such as MCMC must be employed (see [4]).

8.9. Regression That Is Robust to Outliers

A fact of experimental life is that if you can measure an attribute you can also measure it incorrectly. Despite the increase in fidelity of survey data sets, any regression or model fitting must be able to account for outliers from the fit. For the standard least-squares regression the use of an L_2 norm results in outliers that have substantial leverage in any fit (contributing as the square of the systematic deviation). If we knew

$e(y_i|y)$ for all of the points in our sample (e.g., they are described by an exponential distribution where we would use the L_1 norm to define the error) then we would simply include the error distribution when defining the likelihood. When we do not have a priori knowledge of $e(y_i|y)$, things become more difficult. We can either model $e(y_i|y)$ as a mixture model (see § 5.6.7) or assume a form for $e(y_i|y)$ that is less sensitive to outliers. An example of the latter would be the adoption of the L_1 norm, $\sum_i ||y_i - w_i x_i||$, which we introduced in § 8.3, which is less sensitive to outliers than the L_2 norm (and was, in fact, proposed by Rudjer Bošković prior to the development of least-squares regression by Legendre, Gauss, and others [2]). Minimizing the L_1 norm is essentially finding the median. The drawback of this least absolute value regression is that there is no closed-form solution and we must minimize the likelihood space using an iterative approach.

Other approaches to robust regression adopt an approach that seeks to reject outliers. In the astronomical community this is usually referred to as "sigma clipping" and is undertaken in an iterative manner by progressively pruning data points that are not well represented by the model. Least-trimmed squares formalizes this, somewhat ad hoc approach, by searching for the subset of K points which minimize $\sum_i^K (y_i - \theta_i x_i)^2$. For large N the number of combinations makes this search expensive.

Complementary to outlier rejection are the Theil–Sen [20] or the Kendall robust line-fit method and associated techniques. In these cases the regression is determined from the median of the slope, θ_1, calculated from all pairs of points within the data set. Given the slope, the offset or zero point, θ_0, can be defined from the median of $y_i - \theta_1 x_i$. Each of these techniques is simple to estimate and scales to large numbers.

M estimators (M stands for "maximum-likelihood-type") approach the problem of outliers by modifying the underlying likelihood estimator to be less sensitive than the classic L_2 norm. M estimators are a class of estimators that include many maximum-likelihood approaches (including least squares). They replace the standard least squares, which minimizes the sum of the squares of the residuals between a data value and the model, with a different function. Ideally the M estimator has the property that it increases less than the square of the residual and has a unique minimum at zero.

Huber loss function

An example of an M estimator that is common in robust regression is that of the Huber loss (or cost) function [9]. The Huber estimator minimizes

$$\sum_{i=1}^{N} e(y_i|y), \tag{8.65}$$

where $e(y_i|y)$ is modeled as

$$\phi(t) = \begin{cases} \frac{1}{2}t^2 & \text{if } |t| \leq c, \\ c|t| - \frac{1}{2}c^2 & \text{if } |t| \geq c, \end{cases} \tag{8.66}$$

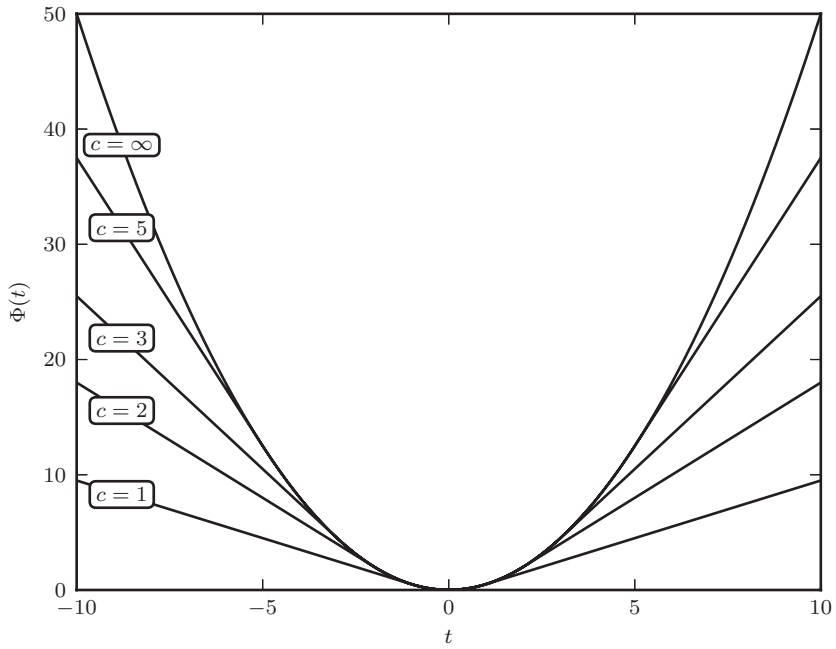


Figure 8.7. The Huber loss function for various values of c.

and $t = y_i - y$ with a constant c that must be chosen. Therefore, $e(t)$ is a function which acts like t^2 for $|t| \leq c$ and like $|t|$ for $|t| > c$ and is continuous and differentiable (see figure 8.7). The transition in the Huber function is equivalent to assuming a Gaussian error distribution for small excursions from the true value of the function and an exponential distribution for large excursions (its behavior is a compromise between the mean and the median). Figure 8.8 shows an application of the Huber loss function to data with outliers. Outliers do have a small effect, and the slope of the Huber loss fit is closer to that of standard linear regression.

8.9.1. Bayesian Outlier Methods

From a Bayesian perspective, one can use the techniques developed in chapter 5 within the context of a regression model in order to account for, and even to individually identify outliers (recall § 5.6.7). Figure 8.9 again shows the data set used in figure 8.8, which contains three clear outliers. In a standard straight-line fit to the data, the result is strongly affected by these points. Though this standard linear regression problem is solvable in closed form (as it is in figure 8.8), here we compute the best-fit slope and intercept using MCMC sampling (and show the resulting contours in the upper-right panel).

The remaining two panels show two different Bayesian strategies for accounting for outliers. The main idea is to enhance the model such that it can naturally explain the presence of outliers. In the first model, we account for the outliers through the use of a mixture model, adding a background Gaussian component to our data. This is the regression analog of the model explored in § 5.6.5, with the difference that here we are modeling the background as a wide Gaussian rather than a uniform distribution.

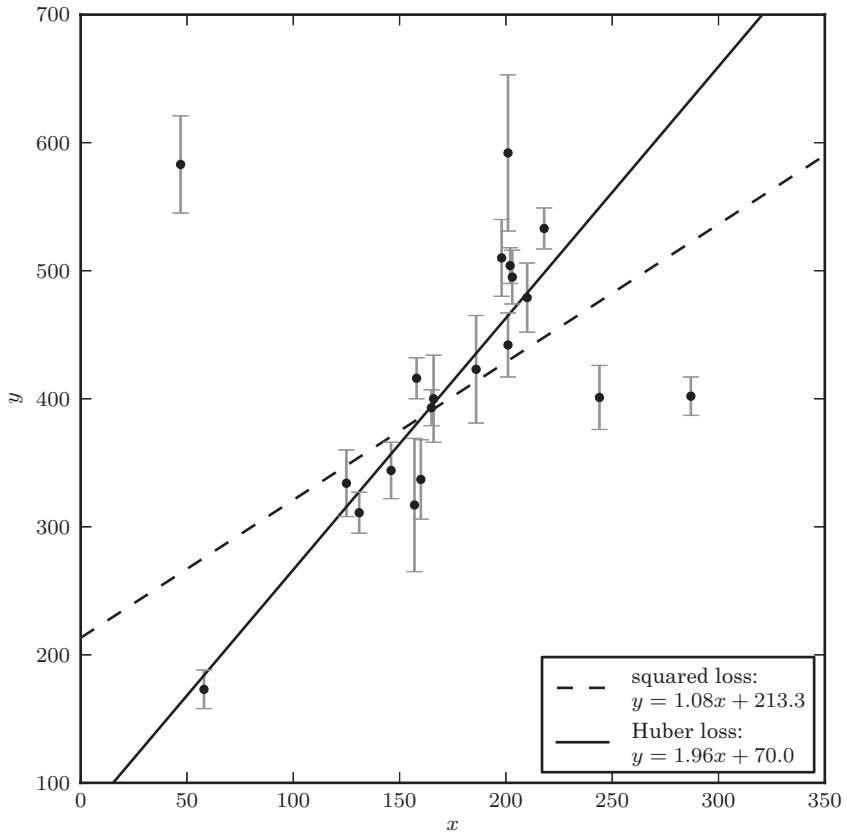


Figure 8.8. An example of fitting a simple linear model to data which includes outliers (data is from table 1 of [8]). A comparison of linear regression using the squared-loss function (equivalent to ordinary least-squares regression) and the Huber loss function, with $c = 1$ (i.e., beyond 1 standard deviation, the loss becomes linear).

The mixture model includes three additional parameters: μ_b and V_b, the mean and variance of the background, and p_b, the probability that any point is an outlier. With this model, the likelihood becomes (cf. eq. 5.83; see also [8])

$$p(\{y_i\}|\{x_i\}, \{\sigma_i\}, \theta_0, \theta_1, \mu_b, V_b, p_b) \propto \prod_{i=1}^{N} \left[\frac{1-p_b}{\sqrt{2\pi\sigma_i^2}} \exp\left(-\frac{(y_i - \theta_1 x_i - \theta_0)^2}{2\sigma_i^2} \right) + \frac{p_b}{\sqrt{2\pi(V_b + \sigma_i^2)}} \exp\left(-\frac{(y_i - \mu_b)^2}{2(V_b + \sigma_i^2)} \right) \right]. \tag{8.67}$$

Using MCMC sampling and marginalizing over the background parameters yields the dashed-line fit in figure 8.9. The marginalized posterior for this model is shown in the lower-left panel. This fit is much less affected by the outliers than is the simple regression model used above.

Finally, we can go further and perform an analysis analogous to that of § 5.6.7, in which we attempt to identify bad points individually. In analogy with eq. 5.94 we

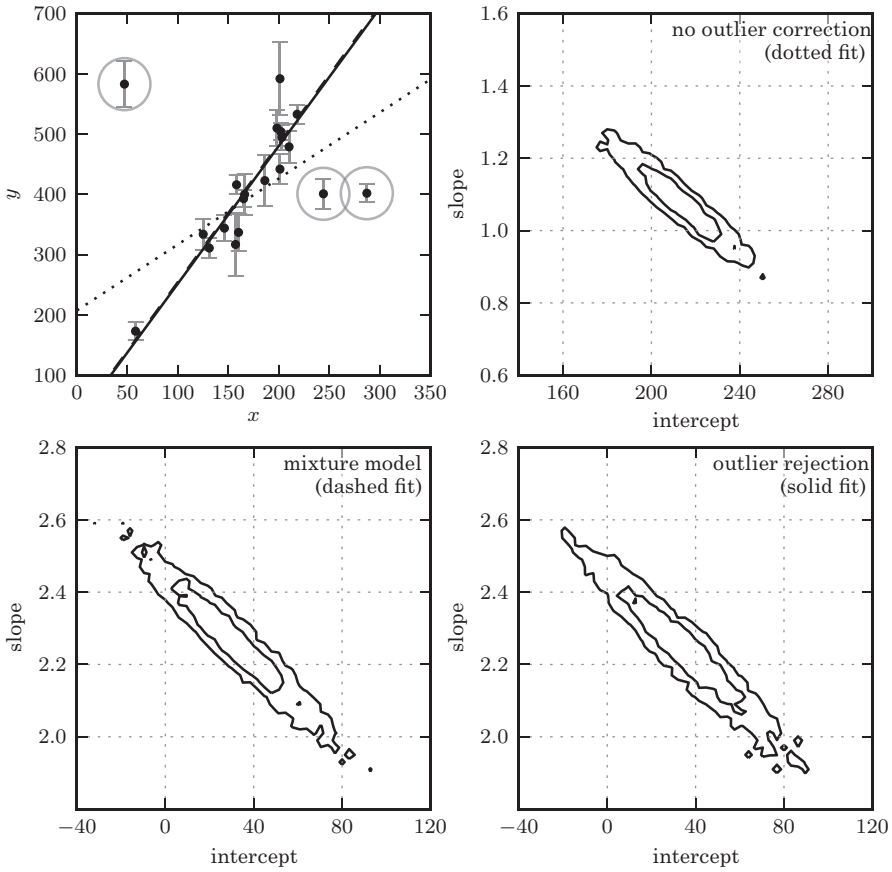


Figure 8.9. Bayesian outlier detection for the same data as shown in figure 8.8. The top-left panel shows the data, with the fits from each model. The top-right panel shows the 1σ and 2σ contours for the slope and intercept with no outlier correction: the resulting fit (shown by the dotted line) is clearly highly affected by the presence of outliers. The bottom-left panel shows the marginalized 1σ and 2σ contours for a mixture model (eq. 8.67). The bottom-right panel shows the marginalized 1σ and 2σ contours for a model in which points are identified individually as "good" or "bad" (eq. 8.68). The points which are identified by this method as bad with a probability greater than 68% are circled in the first panel.

can fit for nuisance parameters g_i, such that if $g_i = 1$, the point is a "good" point, and if $g_i = 0$ the point is a "bad" point. With this addition our model becomes

$$p(\{y_i\}|\{x_i\}, \{\sigma_i\}, \{g_i\}, \theta_0, \theta_1, \mu_b, V_b) \propto \prod_{i=1}^{N} \left[\frac{g_i}{\sqrt{2\pi\sigma_i^2}} \exp\left(-\frac{(y_i - \theta_1 x_i - \theta_0)^2}{2\sigma_i^2}\right) + \frac{1 - g_i}{\sqrt{2\pi(V_b + \sigma_i^2)}} \exp\left(-\frac{(y_i - \mu_b)^2}{2(V_b + \sigma_i^2)}\right) \right]. \tag{8.68}$$

This model is very powerful: by marginalizing over all parameters but a particular g_i, we obtain a posterior estimate of whether point i is an outlier. Using this procedure,

the "bad" points have been marked with a circle in the upper-left panel of figure 8.9. If instead we marginalize over the nuisance parameters, we can compute a two-dimensional posterior for the slope and intercept of the straight-line fit. The lower-right panel shows this posterior, after marginalizing over $\{g_i\}$, μ_b, and V_b. In this example, the result is largely consistent with the simpler Bayesian mixture model used above.

8.10. Gaussian Process Regression

Another powerful class of regression algorithms is Gaussian process regression. Despite its name, Gaussian process regression is widely applicable to data that are not generated by a Gaussian process, and can lead to very flexible regression models that are more data driven than other parametric approaches. There is a rich literature on the subject, and we will only give a cursory treatment here. An excellent book-length treatment of Gaussian processes can be found in [19].

A Gaussian process is a collection of random variables in parameter space, any subset of which is defined by a joint Gaussian distribution. It can be shown that a Gaussian process can be completely specified by its mean and covariance function. Gaussian processes can be defined in any number of dimensions for any positive covariance function. For simplicity, in this section we will consider one dimension and use a familiar squared-exponential covariance function,

$$\mathrm{Cov}(x_1, x_2; h) = \exp\left(\frac{-(x_1 - x_2)^2}{2h^2}\right), \tag{8.69}$$

where h is the bandwidth. Given a chosen value of h, this covariance function specifies the statistics of an infinite set of possible functions $f(x)$. The upper-left panel of figure 8.10 shows three of the possible functions drawn from a zero-mean Gaussian process[3] with $h = 1.0$. This becomes more interesting when we specify constraints on the Gaussian process: that is, we select only those functions $f(x)$ which pass through particular points in the space. The remaining panels of figure 8.10 show this Gaussian process constrained by points without error (upper-right panel), points with error (lower-left panel), and a set of 20 noisy observations drawn from the function $f_{\mathrm{true}}(x) = \cos x$. In each, the shaded region shows the 2σ contour in which 95% of all possible functions $f(x)$ lie.

These constrained Gaussian process examples hint at how these ideas could be used for standard regression tasks. The Gaussian process regression problem can be formulated similarly to the other regression problems discussed above. We assume our data is drawn from an underlying model $f(x)$: that is, our observed data is $\{x_i, y_i = f(x_i) + \sigma_i\}$. Given the observed data, we desire an estimate of the mean value $\bar{f}_j^*$ and variance Σ_{jk}^* for a new set of measurements x_j^*. In Bayesian terms, we want to compute the posterior pdf

$$p(f_j | \{x_i, y_i, \sigma_i\}, x_j^*). \tag{8.70}$$

[3]These functions are drawn by explicitly creating the $N \times N$ covariance matrix C from the functional form in eq. 8.69, and drawing N correlated Gaussian random variables with mean 0 and covariance C.

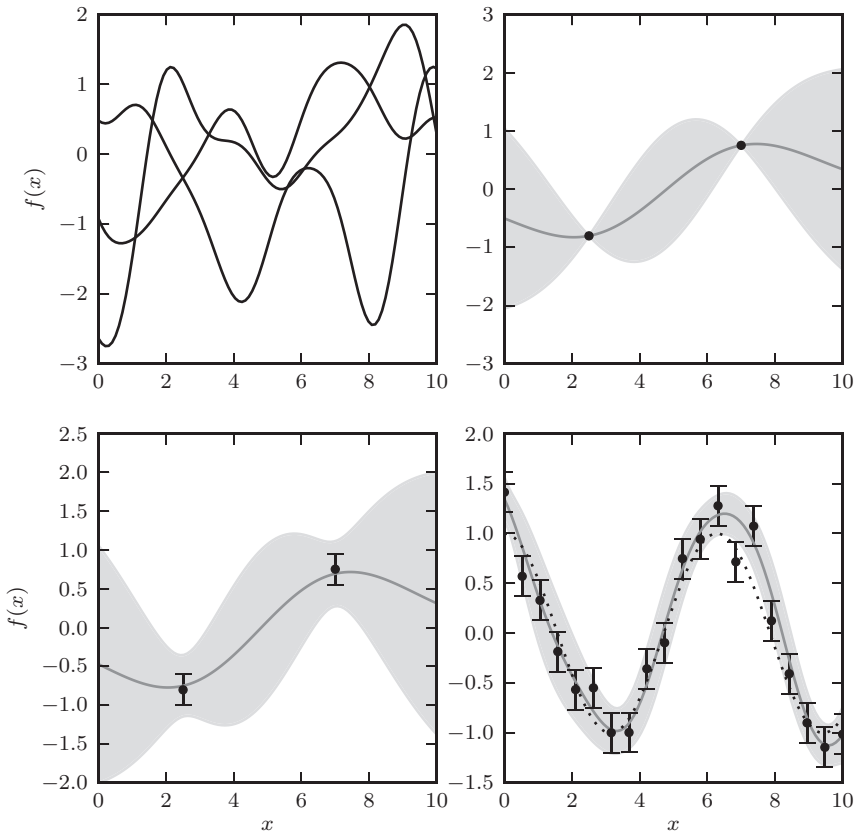


Figure 8.10. An example of Gaussian process regression. The upper-left panel shows three functions drawn from an unconstrained Gaussian process with squared-exponential covariance of bandwidth $h = 1.0$. The upper-right panel adds two constraints, and shows the 2σ contours of the constrained function space. The lower-left panel shows the function space constrained by the points with error bars. The lower-right panel shows the function space constrained by 20 noisy points drawn from $f(x) = \cos x$.

This amounts to averaging over the *entire set* of possible functions $f(x)$ which pass through our constraints. This seemingly infinite calculation can be made tractable using a "kernel trick," as outlined in [19], transforming from the infinite function space to a finite covariance space. The result of this mathematical exercise is that the Gaussian process regression problem can be formulated as follows.

Using the assumed form of the covariance function (eq. 8.69), we compute the covariance matrix

$$K = \begin{pmatrix} K_{11} & K_{12} \\ K_{12}^T & K_{22} \end{pmatrix}, \tag{8.71}$$

where K_{11} is the covariance between the input points x_i with observational errors σ_i^2 added in quadrature to the diagonal, K_{12} is the cross-covariance between the input points x_i and the unknown points x_j^*, and K_{22} is the covariance between the

unknown points x_j^*. Then for observed vectors $\mathbf{x}$ and $\mathbf{y}$, and a vector of unknown points $\mathbf{x}^*$, it can be shown that the posterior in eq. 8.70 is given by

$$p(f_j|\{x_i, y_i, \sigma_i\}, x_j^*) = \mathcal{N}(\mu, \Sigma), \tag{8.72}$$

where

$$\mu = K_{12}K_{11}^{-1}\mathbf{y}, \tag{8.73}$$

$$\Sigma = K_{22} - K_{12}^T K_{11}^{-1} K_{12}. \tag{8.74}$$

Then μ_j gives the expected value $\bar{f}_j^*$ of the result, and Σ_{jk} gives the error covariance between any two unknown points. Note that the physics of the underlying process enters through the assumed form of the covariance function (e.g., eq. 8.69; for an analysis of several plausible covariance functions in the astronomical context of quasar variability; see [24]).

In figure 8.11, we show a Gaussian process regression analysis of the supernova data set used above. The model is very well constrained near $z = 0.6$, where there is a lot of data, but not well constrained at higher redshifts. This is an important feature of Gaussian process regression: the analysis produces not only a best-fit model, but an uncertainty at each point, as well as a full covariance estimate of the result at unknown points.

A powerful and general framework for Gaussian process regression is available in Scikit-learn. It can be used as follows:

```
import numpy as np
from sklearn.gaussian_process import GaussianProcess

X = np.random.random((100, 2)) # 100 pts in 2 dims
y = np.sin(10 * X[:, 0] + X[:, 1])
gp = GaussianProcess(corr='squared_exponential')
gp.fit(X, y)
y_pred, dy_pred = gp.predict(X, eval_MSE=True)
```

The `predict` method can optionally return the mean square error, which is the diagonal of the covariance matrix of the fit. For more details, see the source code of the figures in this chapter, as well as the Scikit-learn documentation.

The results shown in figure 8.11 depend on correctly tuning the correlation function bandwidth h. If h is too large or too small, the particular Gaussian process model will be unsuitable for the data and the results will be biased. Figure 8.11 uses a simple cross-validation approach to decide on the best value of h: we will discuss cross-validation in the following section. Sometimes, the value of h is scientifically an interesting outcome of the analysis; such an example is discussed in the context of the damped random walk model in § 10.5.4.

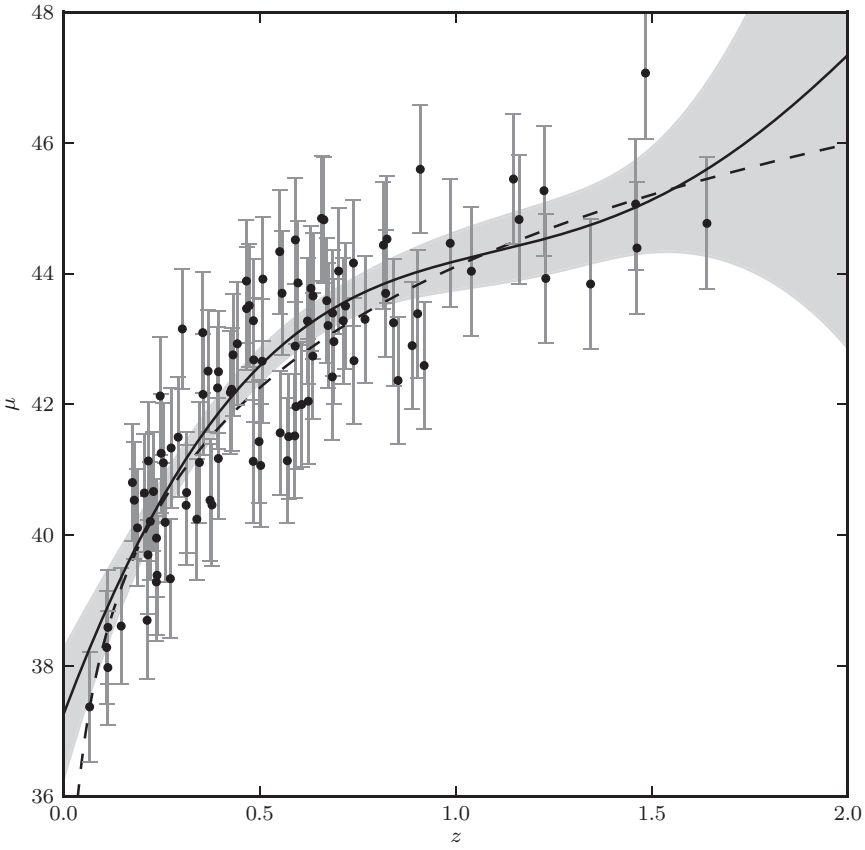


Figure 8.11. A Gaussian process regression analysis of the simulated supernova sample used in figure 8.2. This uses a squared-exponential covariance model, with bandwidth learned through cross-validation.

8.11. Overfitting, Underfitting, and Cross-Validation

When using regression, whether from a Bayesian or maximum likelihood perspective, it is important to recognize some of the potential pitfalls associated with these methods. As noted above, the optimality of the regression is contingent on correct model selection. In this section we explore cross-validation methods which can help determine whether a potential model is a good fit to the data. These techniques are complementary to the model selection techniques such as AIC and BIC discussed in § 4.3. This section will introduce the important topics of overfitting and underfitting, bias and variance, and introduces the frequentist tool of cross-validation to understand these.

Here, for simplicity, we will consider the example of a simple one-dimensional model with homoscedastic errors, though the results of this section naturally generalize to more sophisticated models. As above, our observed data is x_i, and we're trying to predict the dependent variable y_i. We have a training sample in which we have observed both x_i and y_i, and an unknown sample for which only x_i is measured.

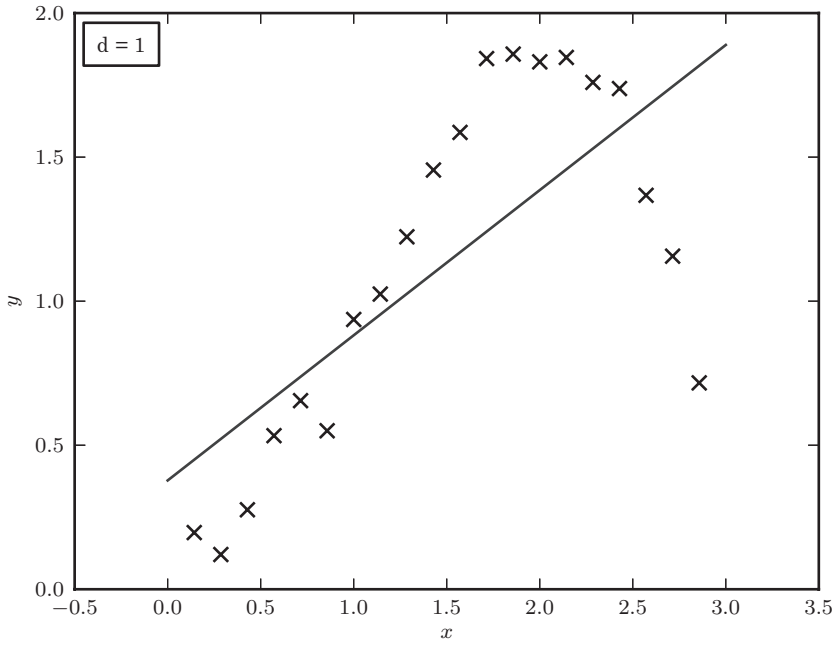


Figure 8.12. Our toy data set described by eq. 8.75. Shown is the line of best fit, which quite clearly underfits the data. In other words, a linear model in this case has high bias.

For example, you may be looking at the fundamental plane for elliptical galaxies, and trying to predict a galaxy's central black hole mass given the velocity dispersion and surface brightness of the stars. Here y_i is the mass of the black hole, and x_i is a vector of a length two consisting of velocity dispersion and surface brightness measurements.

Throughout the rest of this section, we will use a simple model where x and y satisfy the following:

$$\begin{aligned} 0 \leq x_i \leq 3, \\ y_i = x_i \sin(x_i) + \epsilon_i, \end{aligned} \tag{8.75}$$

where the noise is drawn from a normal distribution $\epsilon_i \sim \mathcal{N}(0, 0.1)$. The values for 20 regularly spaced points are shown in figure 8.12.

We will start with a simple straight-line fit to our data. The model is described by two parameters, the slope of the line, θ_1, and the y-axis intercept, θ_0, and is found by minimizing the mean square error,

$$\epsilon = \frac{1}{N} \sum_{i=1}^{N} (y_i - \theta_0 - \theta_1 x_i)^2. \tag{8.76}$$

The resulting best-fit line is shown in figure 8.12. It is clear that a straight line is not a good fit: it does not have enough flexibility to accurately model the data. We say in this case that the model is biased, and that it underfits the data.

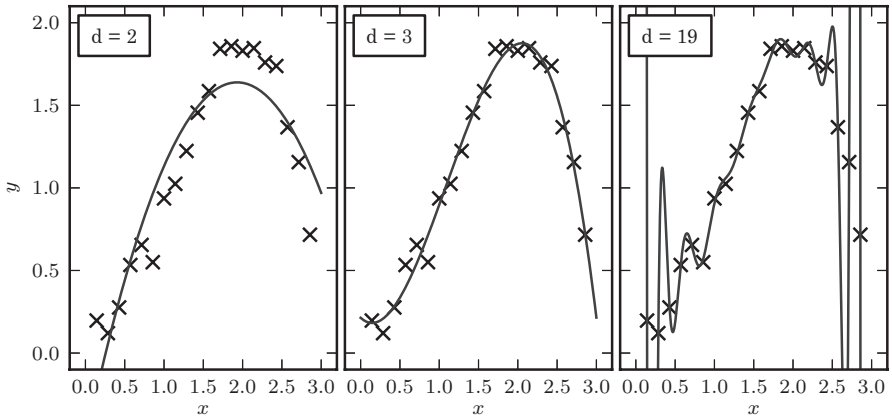


Figure 8.13. Three models of increasing complexity applied to our toy data set (eq. 8.75). The $d = 2$ model, like the linear model in figure 8.12, suffers from high bias, and underfits the data. The $d = 19$ model suffers from high variance, and overfits the data. The $d = 3$ model is a good compromise between these extremes.

What can be done to improve on this? One possibility is to make the model more sophisticated by increasing the degree of the polynomial (see § 8.2.2). For example, we could fit a quadratic function, or a cubic function, or in general a d-degree polynomial. A more complicated model with more free parameters should be able to fit the data much more closely. The panels of figure 8.13 show the best-fit polynomial model for three different choices of the polynomial degree d.

As the degree of the polynomial increases, the best-fit curve matches the data points more and more closely. In the extreme of $d = 19$, we have 20 degrees of freedom with 20 data points, and the training error given by eq. 8.76 can be reduced to zero (though numerical issues can prevent this from being realized in practice). Unfortunately, it is clear that the $d = 19$ polynomial is not a better fit to our data as a whole: the wild swings of the curve in the spaces between the training points are not a good description of the underlying data model. The model suffers from high variance; it overfits the data. The term variance is used here because a small perturbation of one of the training points in the $d = 19$ model can change the best-fit model by a large magnitude. In a high-variance model, the fit varies strongly depending on the exact set or subset of data used to fit it.

The center panel of figure 8.13 shows a $d = 3$ model which balances the trade-off between bias and variance: it does not display the high bias of the $d = 2$ model, and does not display high variance like the $d = 19$ model. For simple two-dimensional data like that seen here, the bias/variance trade-off is easy to visualize by plotting the model along with the input data. But this strategy is not as fruitful as the number of data dimensions grows. What we need is a general measure of the "goodness of fit" of different models to the training data. As displayed above, the mean square error does not paint the whole picture: increasing the degree of the polynomial in this case can lead to smaller and smaller training errors, but this reflects overfitting of the data rather than an improved approximation of the underlying model.

An important practical aspect of regression analysis lies in addressing this deficiency of the training error as an evaluation of goodness of fit, and finding a

model which best compromises between high bias and high variance. To this end, the process of *cross-validation* can be used to quantitatively evaluate the bias and variance of a regression model.

8.11.1. Cross-Validation

There are several possible approaches to cross-validation. We will discuss one approach in detail here, and list some alternative approaches at the end of the section. The simplest approach to cross-validation is to split the training data into three parts: the training set, the cross-validation set, and the test set. As a rule of thumb, the training set should comprise 50–70% of the original training data, while the remainder is divided equally into the cross-validation set and test set.

The training set is used to determine the parameters of a given model (i.e., the optimal values of θ_j for a given choice of d). Using the training set, we evaluate the training error $\epsilon_{\rm tr}$ using eq. 8.76. The cross-validation set is used to evaluate the cross-validation error $\epsilon_{\rm cv}$ of the model, also via eq. 8.76. Because this cross-validation set was not used to construct the fit, the cross-validation error will be large for a high-bias (overfit) model, and better represents the true goodness of fit of the model. With this in mind, the model which minimizes this cross-validation error is likely to be the best model in practice. Once this model is determined, the test error is evaluated using the test set, again via eq. 8.76. This test error gives an estimate of the reliability of the model for an unlabeled data set.

Why do we need a test set as well as a cross-validation set? In one sense, just as the parameters (in this case, θ_j) are learned from the training set, the so-called hyperparameters—those parameters which describe the complexity of the model (in this case, d)—are learned from the cross-validation set. In the same way that the parameters can be overfit to the training data, the hyperparameters can be overfit to the cross-validation data, and the cross-validation error gives an overly optimistic estimate of the performance of the model on an unlabeled data set. The test error is a better representation of the error expected for a new set of data. This is why it is recommended to use both a cross-validation set and a test set in your analysis.

A useful way to use the training error and cross-validation error to evaluate a model is to look at the results graphically. Figure 8.14 shows the training error and cross-validation error for the data in figure 8.13 as a function of the polynomial degree d. For reference, the dotted line indicates the level of intrinsic scatter added to our data.

The broad features of this plot reflect what is generally seen as the complexity of a regression model is increased: for small d, we see that both the training error and cross-validation error are very high. This is the tell-tale indication of a high-bias model, in which the model underfits the data. Because the model does not have enough complexity to describe the intrinsic features of the data, it performs poorly for both the training and cross-validation sets.

For large d, we see that the training error becomes very small (smaller than the intrinsic scatter we added to our data) while the cross-validation error becomes very large. This is the telltale indication of a high-variance model, in which the model overfits the data. Because the model is overly complex, it can match subtle variations in the training set which do not reflect the underlying distribution. Plotting this sort of information is a very straightforward way to settle on a suitable model. Of course,

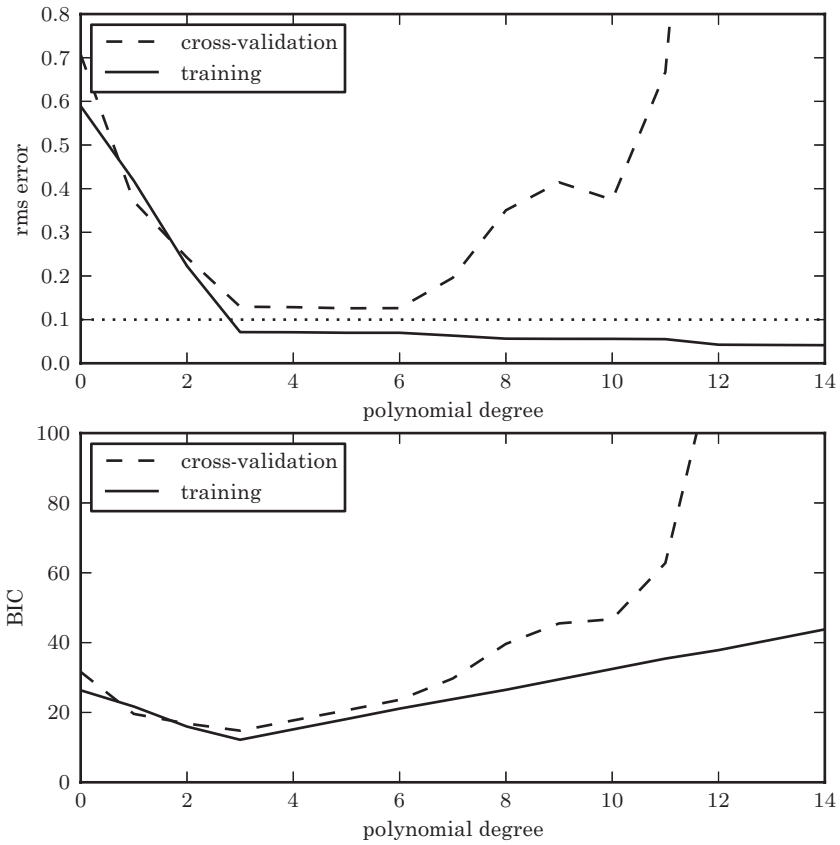


Figure 8.14. The top panel shows the root-mean-square (rms) training error and cross-validation error for our toy model (eq. 8.75) as a function of the polynomial degree d. The horizontal dotted line indicates the level of intrinsic scatter in the data. Models with polynomial degree from 3 to 5 minimize the cross-validation rms error. The bottom panel shows the Bayesian information criterion (BIC) for the training and cross-validation subsamples. According to the BIC, a degree-3 polynomial gives the best fit to this data set.

AIC and BIC provide another way to choose optimal d. Here both methods would choose the model with the best possible cross-validation error: $d = 3$.

8.11.2. Learning Curves

One question that cross-validation does not directly address is that of how to improve a model that is not giving satisfactory results (e.g., the cross-validation error is much larger than the known errors). There are several possibilities:

1. **Get more training data**. Often, using more data to train a model can lead to better results. Surprisingly, though, this is not always the case.
2. **Use a more/less complicated model**. As we saw above, the complexity of a model should be chosen as a balance between bias and variance.
3. **Use more/less regularization**. Including regularization, as we saw in the discussion of ridge regression (see § 8.3.1) and other methods above, can help

with the bias/variance trade-off. In general, increasing regularization has a similar effect to decreasing the model complexity.

4. **Increase the number of features**. Adding more observations of each object in your set can lead to a better fit. But this may not always yield the best results.

The choice of which route to take is far beyond a simple philosophical matter: for example, if you desire to improve your photometric redshifts for a large astronomical survey, it is important to evaluate whether you stand to benefit more from increasing the size of the training set (i.e., gathering spectroscopic redshifts for more galaxies) or from increasing the number of observations of each galaxy (i.e., reobserving the galaxies through other passbands). The answer to this question will inform the allocation of limited, and expensive, telescope time.

Note that this is a fundamentally different question than that explored above. There, we had a fixed data set, and were trying to determine the best model. Here, we assume a fixed model, and are asking how to improve the data set. One way to address this question is by plotting learning curves. A learning curve is the plot of the training and cross-validation error as a function of the number of training points. The details are important, so we will write this out explicitly.

Let our model be represented by the set of parameters $\boldsymbol{\theta}$. In the case of our simple example, $\boldsymbol{\theta} = \{\theta_0, \theta_1, \ldots, \theta_d\}$. We will denote by $\boldsymbol{\theta}^{(n)} = \{\theta_0^{(n)}, \theta_1^{(n)}, \ldots, \theta_d^{(n)}\}$ the model parameters which best fit the first n points of the training data: here $n \leq N_{\rm train}$, where $N_{\rm train}$ is the total number of training points. The truncated training error for $\theta^{(n)}$ is given by

$$\epsilon_{\rm tr}^{(n)} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left[y_i - \sum_{m=0}^{d}\theta_0^{(n)}x_i^m\right]}. \tag{8.77}$$

Note that the training error $\epsilon_{\rm tr}^{(n)}$ is evaluated using only the n points on which the model parameters $\theta^{(n)}$ were trained, not the full set of $N_{\rm train}$ points. Similarly, the truncated cross-validation error is given by

$$\epsilon_{\rm cv}^{(n)} = \sqrt{\frac{1}{n}\sum_{i=1}^{N_{\rm cv}}\left[y_i - \sum_{m=0}^{d}\theta_0^{(n)}x_i^m\right]}, \tag{8.78}$$

where we sum over *all* of the cross-validation points. A learning curve is the plot of the truncated training error and truncated cross-validation error as a function of the size n of the training set used. For our toy example, this plot is shown in figure 8.15 for models with $d = 2$ and $d = 3$. The dotted line in each panel again shows for reference the intrinsic error added to the data.

The two panels show some common features, which are reflective of the features of learning curves for any regression model:

1. As we increase the size of the training set, the training error increases. The reason for this is simple: a model of a given complexity can better fit a small set of data than a large set of data. Moreover, aside from small random fluctuations, we expect this training error to always increase with the size of the training set.

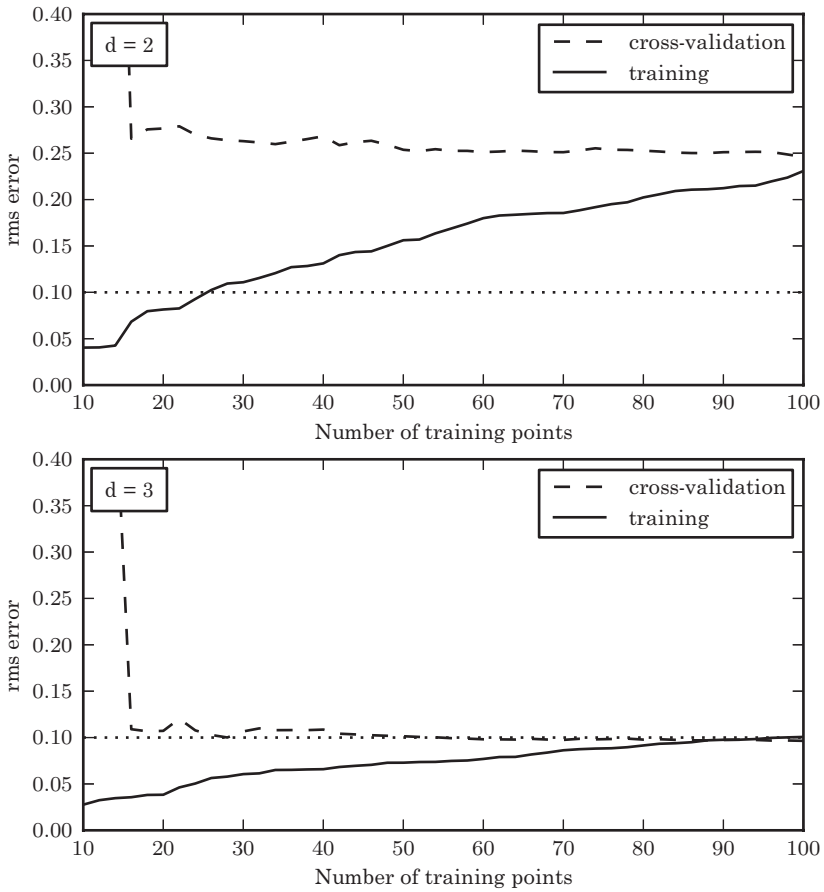


Figure 8.15. The learning curves for the data given by eq. 8.75, with $d = 2$ and $d = 3$. Both models have high variance for a few data points, visible in the spread between training and cross-validation error. As the number of points increases, it is clear that $d = 2$ is a high-bias model which cannot be improved simply by adding training points.

2. As we increase the size of the training set, the cross-validation error decreases. The reason for this is easy to see: a smaller training set leads to overfitting the model, meaning that the model is less representative of the cross-validation data. As the training set grows, overfitting is reduced and the cross-validation error decreases. Again, aside from random fluctuations, and as long as the training set and cross-validation set are statistically similar, we expect the cross-validation error to always decrease as the training set size grows.
3. The training error is everywhere less than or equal to the cross-validation error, up to small statistical fluctuations. We expect the model on average to better describe the data used to train it.
4. The logical outcome of the above three observations is that as the size N of the training set becomes large, the training and cross-validation curves will converge to the same value.

By plotting these learning curves, we can quickly see the effect of adding more training data. When the curves are separated by a large amount, the model error is dominated by variance, and additional training data will help. For example, in the lower panel of figure 8.15, at $N = 15$, the cross-validation error is very high and the training error is very low. Even if we only had 15 points and could not plot the remainder of the curve, we could infer that adding more data would improve the model.

On the other hand, when the two curves have converged to the same value, the model error is dominated by bias, and adding additional training data cannot improve the results *for that model*. For example, in the top panel of figure 8.15, at $N = 100$ the errors have nearly converged. Even without additional data, we can infer that adding data will not decrease the error below about 0.23. Improving the error in this case requires a more sophisticated model, or perhaps more features measured for each point.

To summarize, plotting learning curves can be very useful for evaluating the efficiency of a model and potential paths to improving your data. There are two possible situations:

1. **The training error and cross-validation error have converged**. In this case, increasing the number of training points under the same model is futile: the error cannot improve further. This indicates a model error dominated by bias (i.e., it is underfitting the data). For a high-bias model, the following approaches may help:

    - Add additional features to the data.
    - Increase the model complexity.
    - Decrease the regularization.

2. **The training error is much smaller than the cross-validation error**. In this case, increasing the number of training points is likely to improve the model. This condition indicates that the model error is dominated by variance (i.e., it is overfitting the data). For a high-variance model, the following approaches may help:

    - Increase the training set size.
    - Decrease the model complexity.
    - Increase the amplitude of the regularization.

Finally, we note a few caveats: first, the learning curves seen in figure 8.15 and the model complexity evaluation seen in figure 8.14 are actually aspects of a three-dimensional space. Changing the data and changing the model go hand in hand, and one should always combine the two diagnostics to seek the best match between model and data.

Second, this entire discussion assumes that the training data, cross-validation data, and test data are statistically similar. This analysis will fail if the samples are drawn from different distributions, or have different measurement errors or different observational limits.

8.11.3. Other Cross-Validation Techniques

There are numerous cross-validation techniques available which are suitable for different situations. It is easy to generalize from the above discussion to these various cross-validation strategies, so we will just briefly mention them here.

Twofold cross-validation

Above, we split the data into a training set d_1, a cross-validation set d_2, and a test set d_0. Our simple tests involved training the model on d_0 and cross-validating the model on d_1. In twofold cross-validation, this process is repeated, training the model on d_1 and cross-validating the model on d_0. The training error and cross-validation error are computed from the mean of the errors in each fold. This leads to more robust determination of the cross-validation error for smaller data sets.

K-fold cross-validation

A generalization of twofold cross-validation is K-fold cross-validation. Here we split the data into $K + 1$ sets: the test set d_0, and the cross-validation sets $d_1, d_2, \ldots, d_K$. We train K different models, each time leaving out a single subset to measure the cross-validation error. The final training error and cross-validation error can be computed using the mean or median of the set of results. The median can be a better statistic than the mean in cases where the subsets d_i contain few points.

Leave-one-out cross-validation

At the extreme of K-fold cross-validation is leave-one-out cross-validation. This is essentially the same as K-fold cross-validation, but this time our sets $d_1, d_2, \ldots, d_K$ have only one data point each. That is, we repeatedly train the model, leaving out only a single point to estimate the cross-validation error. Again, the final training error and cross-validation error are estimated using the mean or median of the individual trials. This can be useful when the size of the data set is very small, so that significantly reducing the number of data points leads to much different model characteristics.

Random subset cross-validation

In this approach, the cross-validation set and training set are selected by randomly partitioning the data, and repeating any number of times until the error statistics are well sampled. The disadvantage here is that not every point is guaranteed to be used both for training and for cross-validation. Thus, there is a finite chance that an outlier can lead to spurious results. For N points and P random samplings of the data, this situation becomes very unlikely for $N/2^P \ll 1$.

8.11.4. Summary of Cross-Validation and Learning Curves

In this section we have shown how to evaluate how well a model fits a data set through cross-validation. This is one practical route to the model selection ideas presented in chapters 4–5. We have covered how to determine the best model given a data set (§ 8.11.1) and how to address both the model and the data together to improve results (§ 8.11.2).

Cross-validation is one place where machine learning and data mining may be considered more of an art than a science. The optimal route to improving a model is not always straightforward. We hope that by following the suggestions in this chapter, you can apply this art successfully to your own data.

8.12. Which Regression Method Should I Use?

As we did in the last two chapters, we will use the axes of "accuracy," "interpretability," "simplicity," and "speed" (see § 6.6 for a description of these terms) to provide a rough guide to the trade-offs in choosing between the different regression methods described in this chapter.

What are the most* accurate *regression methods ? The starting point is basic linear regression. Adding ridge or LASSO regularization in principle increases accuracy, since as long as $\lambda = 0$ is among the options tried for λ, it provides a superset of possible complexity trade-offs to be tried. This goes along with the general principle that models with more parameters have a greater chance of fitting the data well. Adding the capability to incorporate measurement errors should in principle increase accuracy, assuming of course that the error estimates are accurate enough. Principal component regression should generally increase accuracy by treating collinearity and effectively denoising the data. All of these methods are of course linear—the largest leap in accuracy is likely to come when going from linear to nonlinear models. A starting point for increasing accuracy through nonlinearity is a linear model on nonlinear transformations of the original data. The extent to which this approach increases accuracy depends on the sensibility of the transformations done, since the nonlinear functions introduced are chosen manually rather than automatically. The sensibility can be diagnosed in various ways, such as checking the Gaussianity of the resulting errors. Truly nonlinear models, in the sense that the variable interactions can be nonlinear as well, should be more powerful in general. The final significant leap of accuracy will generally come from going to nonparametric methods such as kernel regression, starting with Nadaraya–Watson regression and more generally, local polynomial regression. Gaussian process regression is typically even more powerful in principle, as it effectively includes an aspect similar to that of kernel regression through the covariance matrix, but also learns coefficients on each data point.

What are the most* interpretable *regression methods ? Linear methods are easy to interpret in terms of understanding the relative importance of each variable through the coefficients, as well as reasoning about how the model will react to different inputs. Ridge or LASSO regression in some sense increase interpretability by identifying the most important features. Because feature selection happens as the result of an overall optimization, reasoning deeply about why particular features were kept or eliminated is not necessarily fruitful. Bayesian formulations, as in the case of models that incorporate errors, add to interpretability by making assumptions clear. PCR begins to decrease interpretability in the sense that the columns are no longer identifiable in their original terms. Moving to nonlinear methods, generalized linear models maintain the identity of the original columns, while general nonlinear

TABLE 8.1.
A summary of the practical properties of different regression methods.

Method	Accuracy	Interpretability	Simplicity	Speed
Linear regression	L	H	H	H
Linear basis function regression	M	M	M	M
Ridge regression	L	H	M	H
LASSO regression	L	H	M	L
PCA regression	M	M	M	M
Nadaraya–Watson regression	M/H	M	H	L/M
Local linear/polynomial regression	H	M	M	L/M
Nonlinear regression	M	H	L	L/M

models tend to become much less interpretable. Kernel regression methods can arguably be relatively interpretable, as their behavior can be reasoned about in terms of distances between points. Gaussian process regression is fairly opaque, as it is relatively difficult to reason about the behavior of the inverse of the data covariance matrix.

What are the* simplest *regression methods ? Basic linear regression has no tunable parameters, so it qualifies as the simplest out-of-the-box method. Generalized linear regression is similar, aside from the manual preparation of the nonlinear features. Ridge and LASSO regression require that only one parameter is tuned, and the optimization is convex, so that there is no need for random restarts, and cross-validation can make learning fairly automatic. PCR is similar in that there is only one critical parameter and the optimization is convex. Bayesian formulations of regression models which result in MCMC are subject to our comments in § 5.9 regarding fiddliness (as well as computational cost). Nadaraya–Watson regression typically has only one critical parameter, the bandwidth. General local polynomial regression can be regarded as having another parameter, the polynomial order. Basic Gaussian process regression has only one critical parameter, the bandwidth of the covariance kernel, and is convex, though extensions with several additional parameters are often used.

What are the most* scalable *regression methods ? Methods based on linear regression are fairly tractable using state-of-the-art linear algebra methods, including basic linear regression, ridge regression, and generalized linear regression, assuming the dimensionality is not excessively high. LASSO requires the solution of a linear program, which becomes expensive as the dimension rises. For PCR, see our comments in § 8.4 regarding PCA and SVD. Kernel regression methods are naively $\mathcal{O}(N^2)$ but can be sped up by fast tree-based algorithms in ways similar to those discussed in § 2.5.2 and chapter 6. Certain approximate algorithms exist for Gaussian process regression, which is naively $\mathcal{O}(N^3)$, but GP is by far the most expensive among the methods we have discussed, and difficult to speed up algorithmically while maintaining high predictive accuracy.

Other considerations, and taste Linear methods can be straightforwardly augmented to handle missing values. The Bayesian approach to linear regression

incorporates measurement errors, while standard versions of machine learning methods do not incorporate errors. Gaussian process regression typically comes with posterior uncertainty bands, though confidence bands can be obtained for any method, as we have discussed in previous chapters. Regarding taste, LASSO is embedded in much recent work on sparsity and is related to work on compressed sensing, and Gaussian process regression has interesting interpretations in terms of priors in function space and in terms of kernelized linear regression.

Simple summary . We summarize our discussion in table 8.1, in terms of "accuracy," "interpretability," "simplicity," and "speed" (see § 6.6 for a description of these terms), given in simple terms of high (H), medium (M), and low (L).

References

[1] Astier, P., J. Guy, N. Regnault, and others (2005). The supernova legacy survey: Measurement of ω_m, ω_λ and w from the first year data set. *Astronomy & Astrophysics 447*(1), 24.

[2] Boscovich, R. J. (1757). De litteraria expeditione per pontificiam ditionem, et synopsis amplioris operis, ac habentur plura ejus ex exemplaria etiam sensorum impressa. *Bononiensi Scientarum et Artum Instituto Atque Academia Commentarii IV*, 353–396.

[3] Cleveland, W. S. (1979). Robust locally weighted regression and smoothing scatterplots. *Journal of the American Statistical Association 74*(368), 829–836.

[4] Dellaportas, P. and D. A. Stephens (1995). Bayesian analysis of errors-in-variables regression models. *Biometrics 51*(3), 1085–1095.

[5] Efron, B., T. Hastie, I. Johnstone, and R. Tibshirani (2004). Least angle regression. *Annals of Statistics 32*(2), 407–451.

[6] Gauss, K. F. (1809). *Theoria Motus Corporum Coelestium in Sectionibus Conicis Solem Ambientium*. Hamburg: Sumtibus F. Perthes et I. H. Besser.

[7] Hogg, D. W., I. K. Baldry, M. R. Blanton, and D. J. Eisenstein (2002). The K correction. *ArXiv:astro-ph/0210394*.

[8] Hogg, D. W., J. Bovy, and D. Lang (2010). Data analysis recipes: Fitting a model to data. *ArXiv:astro-ph/1008.4686*.

[9] Huber, P. J. (1964). Robust estimation of a local parameter. *Annals of Mathematical Statistics 35*, 73–101.

[10] Jolliffe, I. T. (1986). *Principal Component Analysis*. Springer.

[11] Kelly, B. C. (2011). Measurement error models in astronomy. *ArXiv:astro-ph/1112.1745*.

[12] Kim, J., Y. Kim, and Y. Kim (2008). A gradient-based optimization algorithm for LASSO. *Journal of Computational and Graphical Statistics 17*(4), 994–1009.

[13] Krajnović, D. (2011). A Jesuit anglophile: Rogerius Boscovich in England. *Astronomy and Geophysics 52*(6), 060000–6.

[14] Legendre, A. M. (1805). *Nouvelles méthodes pour la détermination des orbites des comètes*. Paris: Courcier.

[15] Levenberg, K. (1944). A method for the solution of certain non-linear problems in least squares. *Quarterly Applied Mathematics II(2)*, 164–168.

[16] Marquardt, D. W. (1963). An algorithm for least-squares estimation of non-linear parameters. *Journal of the Society of Industrial and Applied Mathematics 11*(2), 431–441.

[17] Mertens, B., T. Fearn, and M. Thompson (1995). The efficient cross-validation of principal components applied to principal component regression. *Statistics and Computing 5*, 227–235. 10.1007/BF00142664.

[18] Nadaraya, E. A. (1964). On estimating regression. *Theory of Probability and its Applications 9*, 141–142.

[19] Rasmussen, C. and C. Williams (2005). *Gaussian Processes for Machine Learning*. Adaptive Computation And Machine Learning. MIT Press.

[20] Theil, H. (1950). A rank invariant method of linear and polynomial regression analysis, I, II, III. *Proceedings of the Koninklijke Nederlandse Akademie Wetenschappen, Series A – Mathematical Sciences 53*, 386–392, 521–525, 1397–1412.

[21] Tibshirani, R. J. (1996). Regression shrinkage and selection via the Lasso. *Journal of the Royal Statistical Society, Series B 58*(1), 267–288.

[22] Tikhonov, A. N. (1995). *Numerical Methods for the Solution of Ill-Posed Problems*, Volume 328 of *Mathematics and Its Applications*. Kluwer.

[23] Watson, G. S. (1964). Smooth regression analysis. *Sankhyā Ser. 26*, 359–372.

[24] Zu, Y., C. S. Kochanek, S. Kozłowski, and A. Udalski (2012). Is quasar variability a damped random walk? *ArXiv:astro-ph/1202.3783*.